Requirements Engineering and the Semantic Web

by Scott A. Selberg, Mark Austin

TR 2003-20

Requirements Engineering and the Semantic Web

Scott A. Selberg[*] and Mark A. Austin[†]

April 30, 2003

Abstract

This paper establishes a connection between the field of requirements engineering and the Semantic Web. The basis of the connection comes from identifying both environments as Systems of Systems. The connection brings an ability to apply design strategies and solutions from the Semantic Web to current day requirements engineering problems. This has immediate and long term advantages over current industry tools.

Keywords

Systems Engineering, Requirements Engineering, Semantic Web.

[*]Manufacturing Production Engineer, 1212 Valley House Drive, Agilent Technologies, Rohnert Park, CA 94928, USA. E-mail: scott_selberg@agilent.com

[†]Associate Professor, Department of Civil Engineering and Institute for Systems Research, University of Maryland, College Park, MD 20742, USA. E-mail: austin@isr.umd.edu

Contents

1 Introduction

A fundamental engineering task is understanding a problem or task and then designing a solution. This problem is aggravated by increasingly complex systems developed by teams spread geographically and temporally. It is ironic that the computer, which is intimately connected to any practical solution, is itself a device which is so complex that no individual can completely understand it. In the war on complexity, requirements are an invaluable tool. An unfortunate meta-problem is that the requirements can become so numerous and complex that providing them is no better that handing a person a bag of transistors and asking for a radio back. Thus, not only are requirements needed, but also tools and techniques to apply and manage them. Requirements engineering is a branch of traditional engineering with a focus upon the meta-problem of requirements management.

To manage requirements, formal model-based approaches are needed. Part of the solution involves development of standards for the exchange of requirements (both textual and graphical). See, for example, preliminary documents for AP233 [3, 4, 16, 30]. Another part of the solution involves the development of model-based techniques for connecting the various types of elements contributing to the design. Ramesh and Jarke [31] have developed a traceability reference model which presents a general starting point for developing customized models. Their work highlights the importance of including more than just requirements and design components. In particular, compliance verification procedures (e.g. tests, simulations...), and decisions were identified as critical traceability nodes.

With the standards and models in place, the possibility of developing inference services, such as logical search and automatic validation, opens up. These services are growing in importance because engineering endeavors are expanding in scope and scale, while the development time dwindles. In dealings with the Environmental Protection Agency (EPA) 200,000+ evolving requirements must be constantly monitored[33]. Government projects, such as Space Station Freedom, can have over 1.5 million requirements[5]. The only economically feasible approach to managing this complexity is through automation.

The quest for inference services, which automate tasks, is a current focus of Internet research. The Semantic Web is the title given to the future of the Internet where these services are supported. This common need between the Internet and Requirements Engineering is not a coincidence. Maier and Rechtin proposed a definition of virtual, or chaotic, system of systems[23]. Both the Internet and Requirements Engineering systems fall into this classification. This commonality suggests that the Semantic Web may be a source of solutions to problems faced by requirements engineering.

2 Requirements Engineering

2.1 Definition

Requirements engineering is a branch of traditional engineering with a focus upon the meta-problem of requirements management. It starts with the elicitation of requirements from various stakeholders. During the iterative design cycle, the requirements engineering activities blend into the design activities. The design process makes decisions and specifications to satisfy the requirements while the requirements engineering process tracks and links the specifications to the appropriate requirements.

Within requirements engineering, the terms specifications and requirements are sometimes used interchangeably; however, there is a subtle, but important difference between them. In any engineering effort, there is a need, and something designed to meet that need. The written descriptions of the need are the requirements (e.g., The system shall. . .). The written descriptions of the "thing" are the specifications (e.g., The system can. . .). The choice of 'requirement' or 'specification' reveals the perspective of the speaker. For example a requirement to the manufacturing process becomes a specification to the consumer. Within the iterative design cycle, there is a linkage between requirements and specifications. First, engineers create a specification of the design model to satisfy the requirements. Often the requirements must be extended to account for needs of included modules, which leads to further design work. For example, a specification of 2.4 volt memory requires a 2.4 volt power supply. When the design process is complete, a finished product can be made from the specification. This description of the engineering process is illustrated in Figure 1.

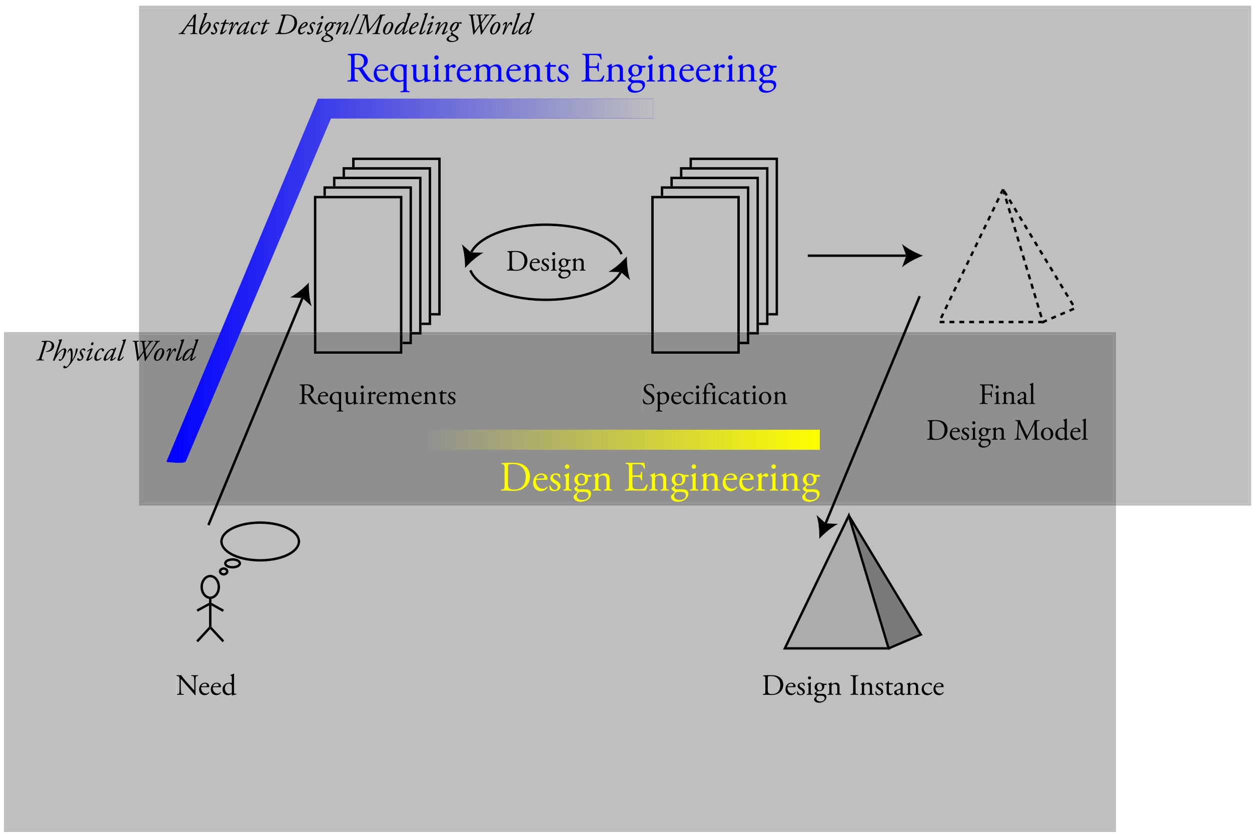


Figure 1: Pathway of Design Development

The origin of Requirements Engineering comes from an obligation to demonstrate that a design satisfies the customer requirements. The mapping of customer requirements to design features is called traceability. Growing awareness of the system life-cycle costs, especially in the maintenance and operation phases, have demonstrated significant benefits from traceability beyond the developer/customer hand-off. Suppose, for example, that a particular requirement is not met – traceability relations help an engineer identify those parts of a system that might be adjusted to correct the deficiency. Traceability can also be applied to the sequence of decisions defining the underlying development process. These paths are of immense value in large projects.

Another reason to employ requirements engineering comes from practical experience "fluctuating and conflicting requirements" [6] have been observed to be a leading difficulty (and cause of failure) in systems engineering projects. Requirements engineering helps to stabilize the requirements documents.

2.2 Requirements Engineering Work Breakdown Structure

Figure 2 presents a work breakdown structure of the requirements engineering process. There are two primary aspects to the requirements engineering activity; creating and using the requirements.

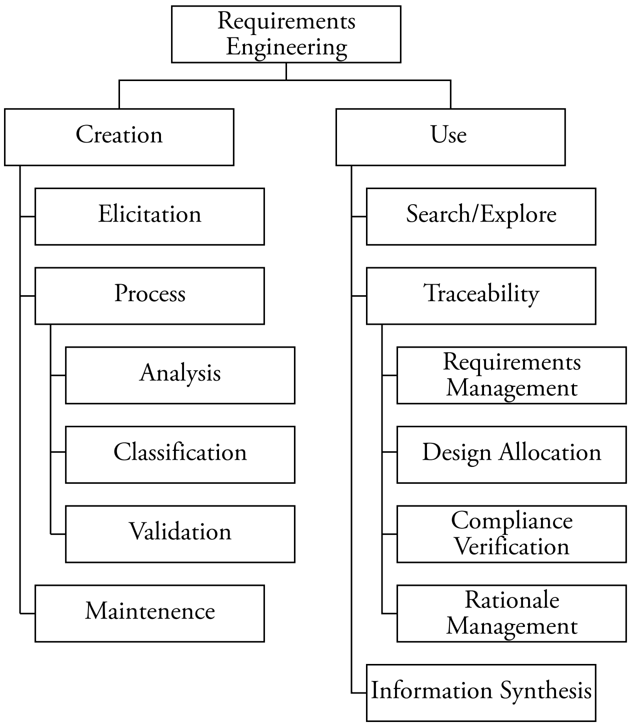


Figure 2: Requirements Engineering Work Breakdown Structure

Creation Branch of Requirements Engineering. The first activity in the process is capturing, qualitatively and quantitatively, the design need from the various stakeholders. This effort is often called elicitation. While requirements elicitation is an easy process to describe, it is a rather difficult process to implement. It does not lend itself easily toward automation, although there are methods and tools to support the process. The major barrier to overcome is communication. Those who have, or perceive, the need, must convey enough of that knowledge to the designers to enable the design process. This is particularly problematic in communities where the vocabulary and experience of the users may be completely disparate from the vocabulary and experience of the designers [11]. As requirements are elicited from the various stakeholders, they need to be processed into a single coherent and consistent form. Each requirement needs to be analyzed for conflicts and ambiguities. The conflicts need to be negotiated with the stakeholders. Ambiguities are resolved through the elicitation of finer grained, derived requirements. The requirements then need to be classified. There are a tremendous number of classification schemes which could be used. The choice of a requirements classification model is very important because, intentional or not, the model places fundamental constraints on the ensuing search, traceability, and synthesis functions. Before the design phase can begin, the initial requirements document needs to be validated against the design need for correctness and completeness. Maintenance is a rather muted term for the last activity contributing to the creation of the requirements documents. As mentioned earlier, the design cycle is iterative and generates requirements. The requirements system needs to capture this information in real time.

Two significant challenges for the requirements system to overcome are time and quality. Capturing requirements is a documentation process. While documentation is be universally accepted as good and worthwhile, it is never the critical path and will be minimized, deferred, or dropped due to unrealistically aggressive project schedules. In cases where time is made for documentation, the quality may still be poor. Some engineers have the skills to transfer their knowledge into writing, others have such poor communication skills that the documentation is misleading at best. These challenges highlight the importance of human factors considerations in the implementation of any requirements system as the system must integrate with the design process without intruding into it. If the system exists on a single computer in a shared office, it will never be used because nobody will have time for it. If it exists as a standalone application in a network environment, for example a web page, it may get some use, but will again be limited by the effort required to stop the current activity and start the documentation activity. As the system becomes more active and starts to integrate itself with the design tools it becomes capable of capturing a complete set of information; however, intrusion becomes an issue. In conclusion the system must take an active roll and prod the designers to insure completeness and quality; however, it must also infer as much information as possible, whenever possible, to remain unobtrusive.

Use Branch of Requirements Engineering. The requirements document is used in three primary ways: (1) It is searched and explored for information, (2) It supports traceability, and (3) It supports information synthesis and inference engines. There are significant challenges facing the implementation of models and tools for each of these activities.

With respect to search and exploration, for example, it is not clear what complexity of search mechanisms should be supported. For a text book, a simple word index at the back of the book is often sufficient. The Internet has pushed this keyword model to its limits with the various search engines such as Metacrawler [26] or Google [13]. A fundamental limitation of this approach is the dependence on keywords. For example, I may be looking for an E4500. Of course, being an ordinary person I know this device as a Nikon Coolpix 4500 digital camera. Or I might be an internal employee, in which case I know the it by some the project name such as, Torpedo. In either case, a keyword search is not going to provide me with a reference to the E4500. There are two approaches to solving this problem. One approach is to develop search engines which are sophisticated enough to understand language semantics and infer the logical connection between the E4500 and an a digital camera. Another approach, and the one currently employed by the Internet, it to use references. For example, if I search the Internet, I may get a page talking about Nikon camera which has a reference, or link, to the E4500 information I am looking for. Thus, traceability complements the searchability of the requirements document. These two approaches can, and should, be simultaneously employed.

Information synthesis deals with two processes: inference and consequence exploration. Inference is the process of making conclusions based upon a given information set. For example, it is useful to know if the sum of the requirements has a feasible solution. For example, if a team was designing a cello case, the following three requirements define an empty feasible design space because a cello will not fit inside the overhead compartment of an airplane.

1. The case shall have a rigid outer shell.
2. The case shall be able to contain a cello.
3. The case shall fit in the overhead compartment of an airplane

In this example, the null design space is easy to identify; however, in a project with thousands or millions of requirements, the task is very difficult. This is where an inference service, which automates the process, is of immense value. The strong need for automated inference services stems, in part, from the need to constantly monitor and revalidate the system as requirements change. “For example, in the case of environmental regulations and requirements, the typical month includes...200,000+ requirements/regulations that need to be monitored [33].” This task is too large and has too small a return on investment to be implemented with people in an economically sound manner. Consequence exploration is a fancy term for “what if” scenarios. It is an extremely useful technique to change a specification or requirement and analyze the results. For example, an architect may be interested in what code violations a building designed in Maryland would have if it were built in California. Automation of this task not only requires the ability to reason about requirements, but also the ability to interchange large trees of requirements.

2.3 Low- and High-End Models of Traceability

Ramesh and Jarke [31] have recently introduced a pair of traceability reference models for low- and high-end users of traceability. Figure 3 is the reference model for low-end users. Each entity and link in the model needs to be elaborated to create organizational and project specific models.

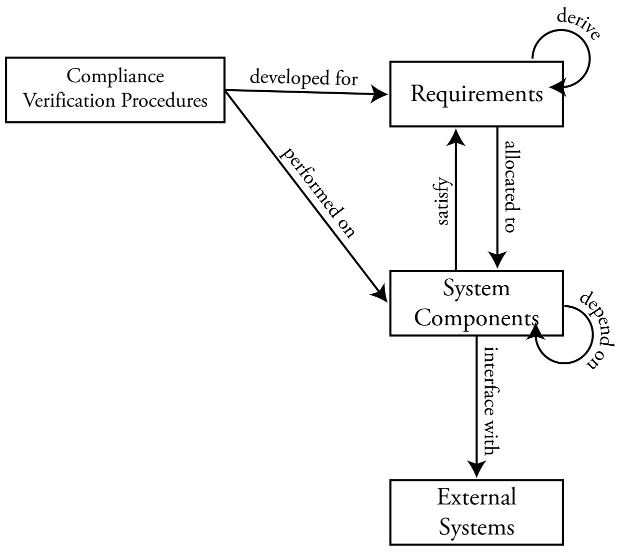


Figure 3: Low-End Traceability Reference Model (Adapted from Ramesh and Jarke [31])

Low End Users of Traceability. Industry experience indicates that low-end users of traceability tend to have problems that require less than about 1,000 requirements (viewed as a mandate from the project sponsors or for compliance with standards). The main applications of traceability for low-end users are: requirements decomposition, requirements allocation, compliance verification, change control. Points to note are as follows:

1. Typical low-end users view requirements traceability as providing a link from requirements to the actual system components that satisfy these requirements. Before this can happen, however, requirements must be derived from higher-level system requirements.

2. Original and derived requirements are allocated to the system components. An "allocation table" is a common mechanism used to maintain this information. This simple two-way mapping between requirements and system components is used to ensure that there are components in the system that satisfy all requirements.

3. In the compliance verification phase of systems development, low-end users employ the requirements database to develop system compliance verification procedures.

Typically, low-end users lack support for capturing rationale for requirements issues and how they are resolved. Similarly, information also tends to be missing from the design and analysis phases of development.

High-End Users of Traceability. Industry experience indicates that high-end users of traceability tend to have problems that require, on average, about 10,000 requirements (viewed as a major opportunity for customer satisfaction and knowledge creation throughout the system lifecycle). They view traceability as an opportunity to increase the probability of producing a system that meets all customer requirements, is easier to maintain, and can be produced within cost and on schedule.

Due to the significant size and complexity of systems under consideration, high-end traceability employs much richer schemes of traceability (e.g., capture of discussion issues, decisions and rationale along product and process-related dimensions) than their low-end counterparts. Figure 4 is an overview diagram of their reference model. Figures 5, 6, 7, and 8 show details of the reference sub-models for compliance verification, design rationale, requirements management, and design allocation. To implement their high-end traceability model, requirements engineering must becomes a full life-cycle activity concerned with more than just requirements.

2.4 State-of-the-Art Capability for Requirements Management Tools

Ramesh and Jarke [31] have conducted an extensive survey of requirements management tools. They offered the four following needs for a requirements management system which are not supported by the present industry toolset. They are mapped onto the work breakdown structure in Figure 9.

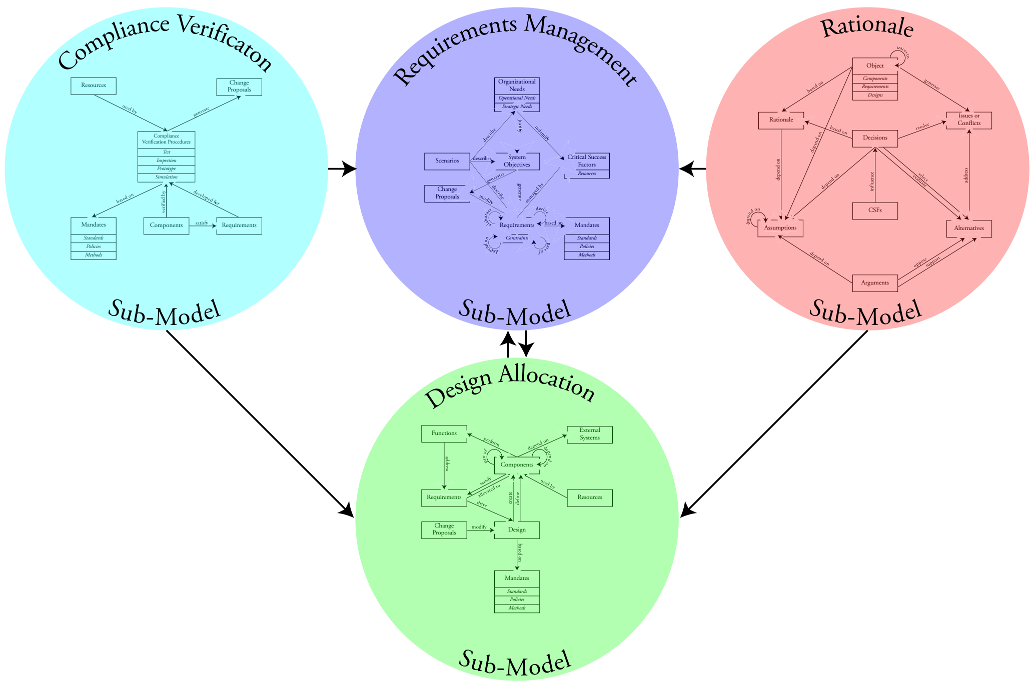


Figure 4: High-End Traceability Reference Model (Adapted from Ramesh and Jarke [31])

1. Thick descriptions
2. Model-driven trace use and capture.
3. Abstraction mechanisms
4. Inference services

1. **Thick Descriptions.** Original sources are often informal and difficult for machines to interpret. Because of this the information is often translated to a more formal, and computer friendly, format. The more formal representations often "result in thin descriptions, with the consequence that much of the meaning embedded in such information is lost. (Ramesh and Jarke, 2001, pg., 88)". Musical notation is a good example. Table 1 compares the traditional musical representation, and a formal computer representation of Siegfried's Short Call by Richard Wagner.

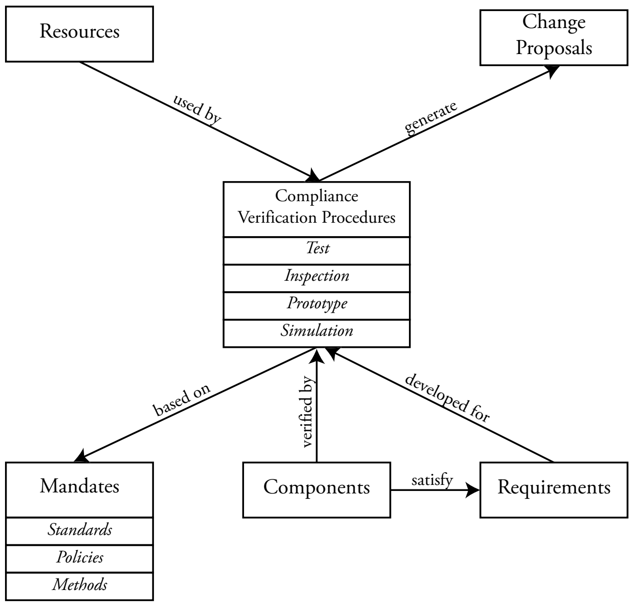


Figure 5: Compliance Verification Sub-Model for High-End Traceability (Adapted from Ramesh and Jarke [31])

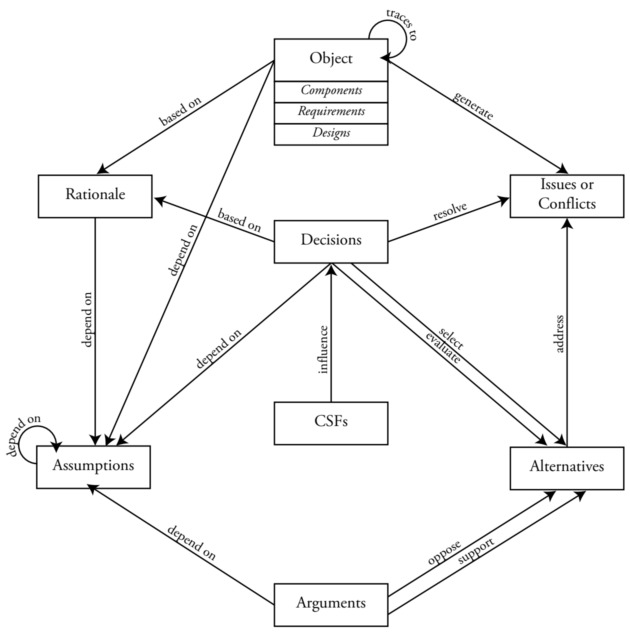


Figure 6: Rationale Sub-Model for High-End Traceability (Adapted from Ramesh and Jarke [31])

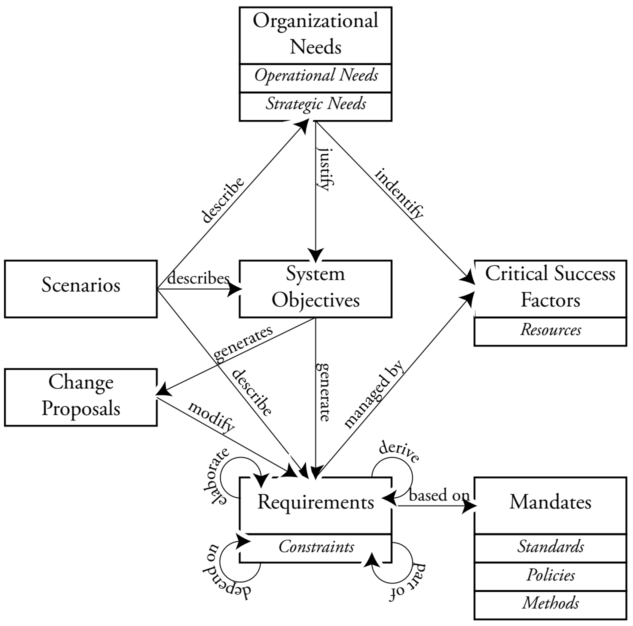


Figure 7: Requirements-Management Sub-Model for High-End Traceability (Adapted from Ramesh and Jarke [31])

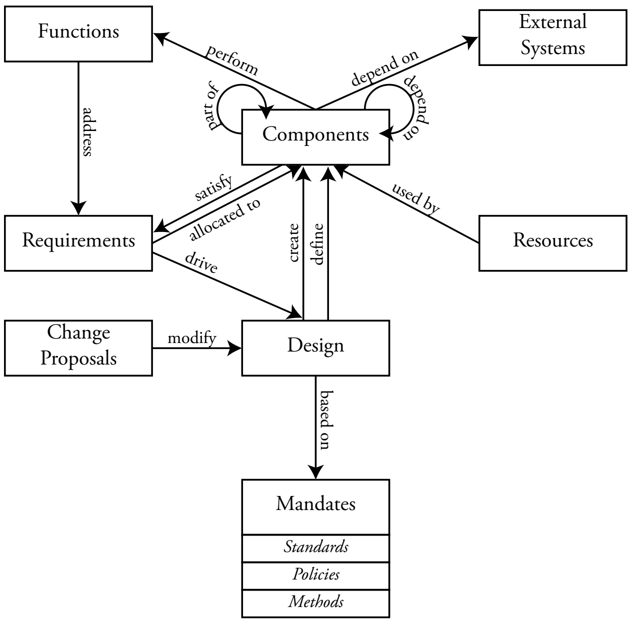


Figure 8: Design Allocation Sub-Model for High-End Traceability (Adapted from Ramesh and Jarke [31])

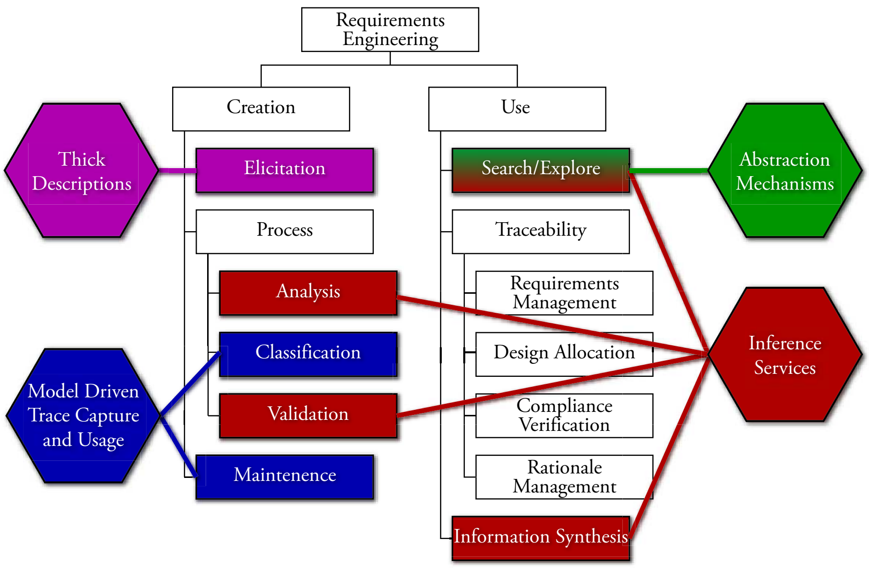


Figure 9: Requirements Engineering Work Breakdown Structure and the Industry Weaknesses identified by Ramesh and Jarke [31]

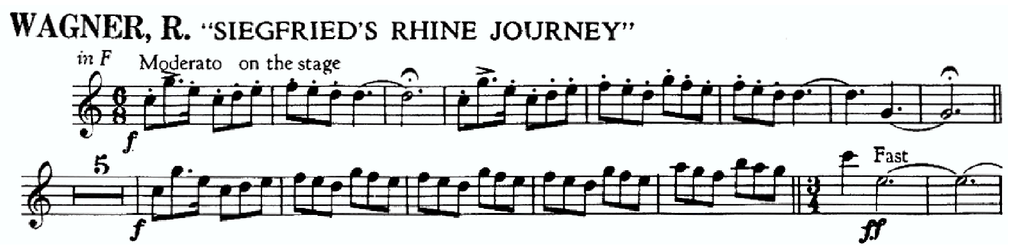
 WAGNER, R. "SIEGFRIED'S RHINE JOURNEY" in F Moderato on the stage f 5 Fast ff Traditional Notation
T150 K1 L32 F O5 C. O4 L64 A L32 F G A B- A G L4 G P64 L32 F O5 C. O4 L64 A L32 F G A B- A G O5 C O4 B- A B- A G L4 G L4 C P64 L32 F O5 C. O4 L64 A L32 F G A B- A G O5 C O4 B- A B- A G O5 C O4 B- A B- A G O5 C O4 B- A O5 D C O4 B- O5 E D C L8 F Computer Notation

Table 2: Comparison of Music in Traditional Notation and Computer Notation

In the translation of the original source to the computer format, some information, such as note articulations, is lost. Other information is collapsed, such as the tempo range of moderato into an exact frequency. The requirements engineering tool needs to have ways of handling the raw "thick" sources as well as the “thin” formal and semi-formal descriptions during the elicitation step. This issue belongs to the elicitation block of the work breakdown structure.

2. **Model Driven Trace Capture and Usage.** There is a need “for mechanisms that allow project managers and developers to define and enact a model-driven trace process (Ramesh and Jarke, 2001, pg., 87).” In other words, the tools today provide no mechanism for defining custom models, such as the previously introduced traceability models. Nor do they provide a mechanism for linking the models into the design process to ensure reliable information collection. These models are an important tool to facilitate the classification of requirements and linking them into the environment during the maintenance activity.

3. **Abstraction Mechanisms.** Methods are needed to provide various views of engineering systems and the requirements documents at various levels of granularity. For example, in a project with several hundred thousand requirements, it is critical to construct a high level view of the project which only contains the few key requirements. It is then important to be able to drill down to a detailed view and examine the fine detailed requirements for any specific sub-system or component. The abstraction mechanisms are an important tool for finding requirements.

4. **Inference Services.** Inference services are a central part of requirements validation and verification (which is part of creation branch of the work breakdown structure) and logical semantic search activities (which is part of use branch of the work breakdown structure).

To date, the industry’s approach to implementation of a requirements management system has been to build a monolithic database system with an open architecture which permits access to external applications. For requirements management alone, this approach is fine; however, it falls short of the needs of a requirements engineering system. The primary difficulty is the ability to leverage or reuse threads and trees of traceability. The vast majority (e.g., 80%) of modern day engineering is routine, meaning that “the underlying procedures and technology are well tested and familiar” [6]. There is thus a strong incentive to reuse work, including requirements. This implies a need to pass requirements between projects, companies, and even industries. In other words, the system will need to operate as a system of systems rather than a large monolithic system.

3 Systems of Systems

In *The Art of Systems Architecting*, Mark Maier and Eberhardt Rechtin define a system of systems as a system in which the constituent systems:

1. Fulfill valid purposes in their own right, and continue to operate to fulfill those purposes if disassembled from the overall system.
2. Are managed (at least in part) for their own purposes rather than the purposes of the whole; the components systems are separately acquired and integrated but maintain a continuing operational existence independent of the collaborative system [23].

This definition can be summarized as a system of systems is a system built from independent and intelligent systems. The primary systems of systems design focus is on the interfaces, specifically the communication standards, as commonly very little can be done to modify the internal workings of the constituent systems. There are three communication styles which influence the design approach: coerced (directed), collaborative, and chaotic (virtual) [22, 23].

1. **Coerced.** In a coerced, or directed, system of systems there is a central entity which can dictate behavior. For example the Windows Operating System dictates certain behaviors of the software applications that run under it.
2. **Collaborative.** A collaborative system of system is composed of several systems working together for synergistic effects. The Adobe product line is a good example. Streamlining the data formats and user interfaces between applications generates increased customer productivity, thus increasing sales. It also locks customers into the Adobe product line, again increasing sales.
3. **Chaotic.** The last management style, chaotic or virtual, has no management structure. The classic example is the Internet. The Internet operates only on recommended standards which the various systems voluntarily adopt. Nothing but public opinion forces applications to adopt the standard, and even if the standard is adopted, nothing guarantees compliance. For example, Cascading Style Sheets have been a official recommendation of the World Wide Web Consortium [40] since 1996; however, even today's browsers are not fully compliant [27]. Chaotic systems of systems work because the various

systems find the interface standards to be advantageous. Companies build web pages which adhere to the HTML standard not because they have to, but because it is advantageous to.

Chaotic systems of systems thrive because of network goods. “Economists call something a “network good” if it increases in value the wider it is consumed. For example, telephones are network goods. A telephone that doesn’t connect to anybody is not valuable. Two cellular telephone networks that can’t inter-operate are much less valuable than if they can inter-operate [23].” Network goods are hard to introduce because they need to achieve a certain magnitude before they become attractive; however, if the hurdle can be overcome, the incentive becomes self driving. A requirements management system, one which is concerned with the creation and use of the requirements for a project, can be implemented as a monolithic system. However, as soon as the need to reuse the requirements across projects, companies, and industries is identified, a monolithic system approach is no longer viable. The chaotic system of systems is the appropriate design model because every project, company, and industry will operate based on personal needs and desires. Figure 10 illustrates how the field of requirements engineering functions as a system of systems.

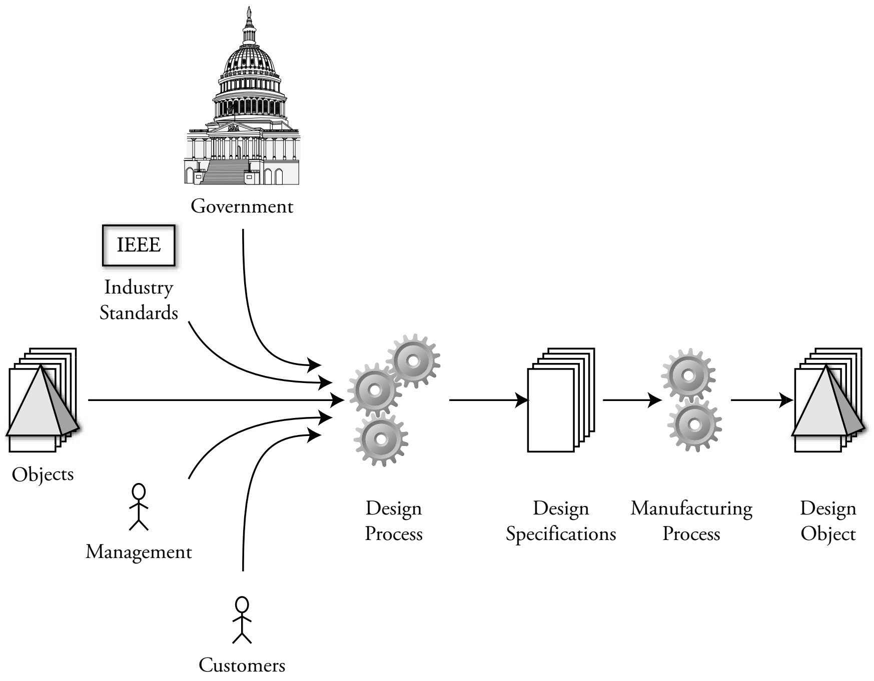


Figure 10: Requirements Engineering as a System of Systems

Since requirements engineering is a system of systems, the appropriate design strategy is to create open standards, which are a network goods. This will allow the various systems to share a common data structures and build customized but compatible tools to meet the personal needs. These are very same tasks faced by the Internet. It is therefore a reasonable approach to compare the needs of a requirements engineering system to the Internet and look for solutions along parallel lines of thought.

4 The Semantic Web

4.1 The Semantic Web Layer Cake

The Semantic Web is the name given to the Internet of the future. Today's web is designed for the presentation of data and information to humans. In contrast, the Semantic Web aims to provide information with a well-defined meaning, thereby creating a pathway for machine-to-machine communication. Automated services will thus be able to perform logical operations based on the semantic descriptions of information on the internet. During a talk at the XML World 2000 Conference in Boston, Massachusetts, the World Wide Web Consortium (W3C) head Tim Berners-Lee presented the Semantic Web Layer Cake diagram (see Figure 11) to describe the infrastructure that will support this vision [8].

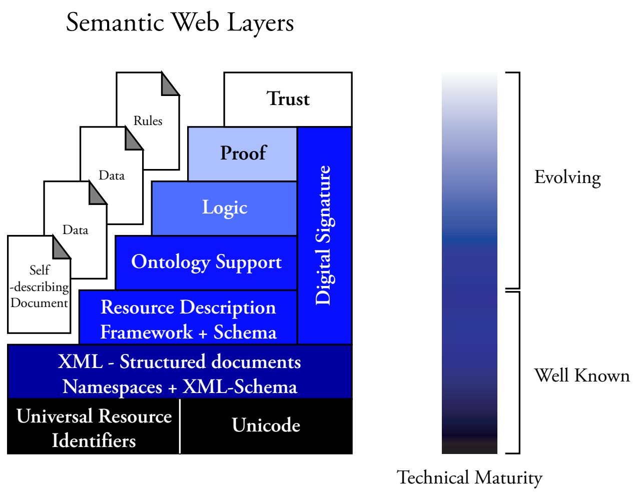

Semantic Web Layers
Rules
Data
Data
Self -describing Document
Trust
Proof
Logic
Ontology Support
Resource Description Framework + Schema
XML - Structured documents Namespaces + XML-Schema
Universal Resource Identifiers
Unicode
Digital Signature
Evolving
Well Known
Technical Maturity

Figure 11: The Semantic Web Layer Cake (Berners-Lee, 2000)

4.1.1 URI and Unicode

The bottom layer of this cake is constructed of Universal Resource Identifiers (URI) [39] and Unicode [38]. URIs are a generalized mechanism for specifying a unique address for an item. They provide the basis for linking information on the Internet. Unicode is the 16-bit extension of ASCII text, which has support for non-english characters. It is the alphabet of the semantic web.

4.1.2 The eXtensible Markup Language (XML)

The next layer up is the eXtensible Markup Language (XML). XML grew out of demands to make HTML more flexible. The technology itself has two aspects. It is an open standard which describes how to encode a tree-based data structure within a plain text file. On a more conceptual plane, XML is a strategy for information management.

The rules of "well-formedness," which are the nuts and bolts part of XML, provide enough information that generic code modules, called parsers, can be developed to read, write, and manipulate the XML files. An example of such a parser is the open source Xerces parser [43]. The parsers can be built into other applications, such as Microsoft Word or Adobe Illustrator [1, 42], giving them the power to work with XML files. The "well-formed" criteria guarantees that the parser can read the XML file, but from the application's point of view, it does not give any confidence that the data in the XML file will be complete or consistent. To solve this problem, the basic form constraint can be extended through the use of Document Type Definitions (DTDs) or Schema. Both of these technologies are ways of specifying a pattern or model to which the XML document must also conform. For example, XHTML, an XML compliant variant of HTML, is defined by both the XML definition and the XHTML DTD [44].

On the conceptual level, XML asks that content be separated from presentation. In this model, presentation is controlled through the use of a style sheet. This information model is extremely powerful because it allows the data to be re-purposed. For example, a single XML file can be presented to the web and paper through two different style sheets. This saves duplication of work and reduces the risk of error.

While XML provides support for the portable encoding of data, it is limited to information that can organized within hierarchical relationships. A common engineering task is the synthesis information from multiple data sources. This can be a problematic situation for XML as a synthesized object may or may not fit into a hierarchial model. Suppose, for example, that within one domain a line is defined by two points, and in a second domain, a point is defined by the intersection of two lines. These definitions and the resulting tree models are illustrated in Figure 12. Merging these models results in a circular reference – the resultant tree model is therefore infinite. XML can not directly support the merger of these two models. A graph, however, can. Thus we introduce the Resource Description Framework.

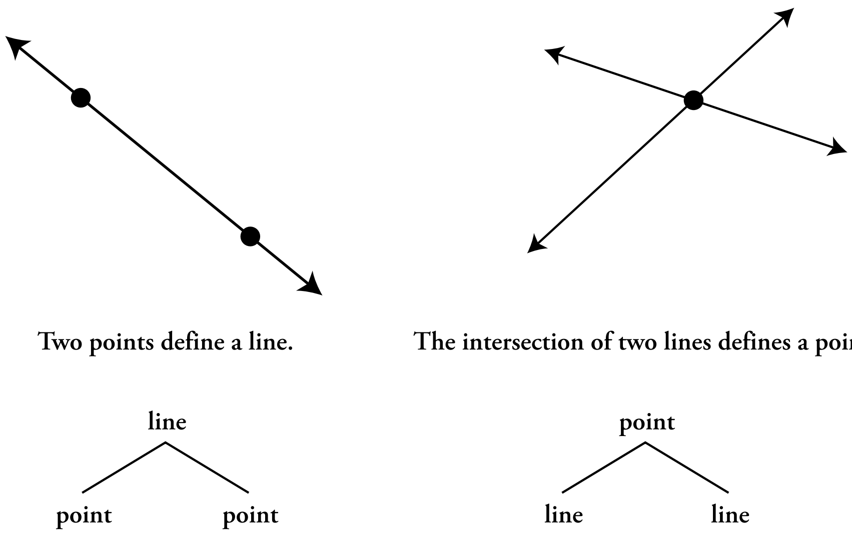

Two points define a line.
The intersection of two lines defines a point.
line
point
point
point
line
line

Figure 12: Definitions of a Line and Point in Tree Model Form

4.1.3 The Resource Description Framework (RDF)

The Resource Description Framework (RDF) defines a standard for describing the relationships between objects and classes in a general but simple way. Within the Semantic Web, for example, a primary use of RDF will be to encode the metadata – information such as the title, author, and subject – associated with web pages. An RDF description is composed as a triplet of subject, property, and object. The property can be an attribute, relationship, or a predicate (verb) [24]. See Figure 13. Some simple examples of RDF are shown in Figure 14.

As with HTML, XML has linking capabilities. The links, via URIs, form the basis for building the

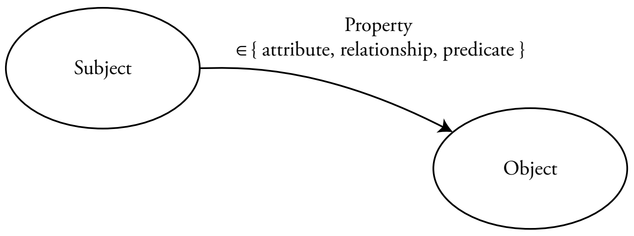


Figure 13: Elements of the RDF triple

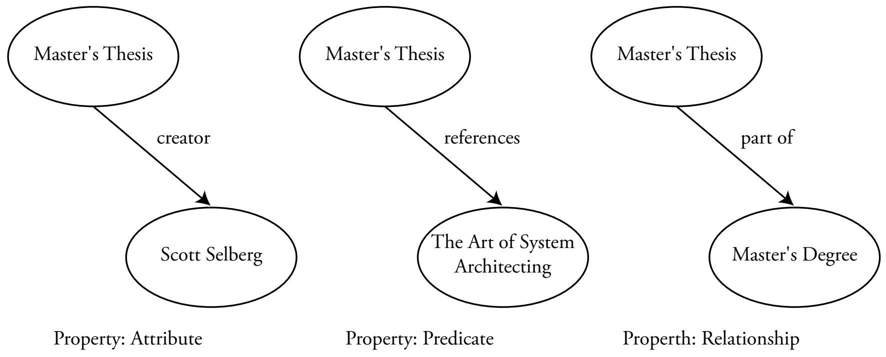


Figure 14: RDF Property Examples

graphs. Thus, RDF can be built upon XML. XML is the bones, RDF is the sinew which ties them together to build a skeleton. Figure 15 shows how the graph-based RDF model can solve the merged tree problem presented in Figure 12.

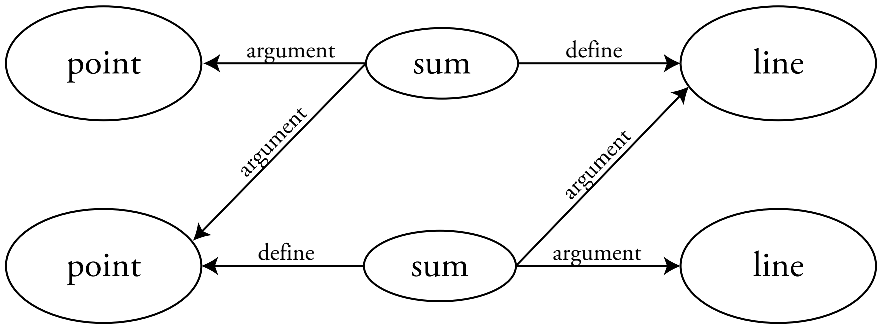


Figure 15: Merging the Definitions with a Graph

As an example of how RDF and XML can be used together, consider a card catalog from a traditional library. Every card has information about a book such as the author, location, and similar subjects. The cards, being much smaller than the books, may be quickly and easily browsed. On the Internet, an XML based web page can have a small, portable RDF description imbedded. These descriptions can be collected to form an internet card catalog.

While linking can be handled in many different schemes, the RDF model presents a tremendous advantage because it's generality allows for the creation and use of standard tools to read, write, and navigate the graphs [32].

A key limitation of RDF is poorly defined semantics. RDF has no sense of vocabulary. It does not provide any notion of scope within which a specified vocabulary can be constrained. Any node within a connected RDF graph is reachable by any other node. To support automated reasoning, agreement on a uniform, well defined, vocabulary is needed.

4.1.4 Ontologies

According to James Hendler, a leading researcher of the Semantic Web, an ontology is "a set of knowledge terms, including the vocabulary, the semantic interconnections, and some simple rules of inference and logic for some particular topic [14, 15]." So what does an ontology look like? It's a question the Internet community is still struggling with. Some envision " a few large, complex, consistent ontologies [15]." Others see "a great number of small ontological components consisting largely of pointers to each other [15]."

While the exact picture of an ontology is unclear, the problems an ontology needs to overcome are agreed upon. They are to facilitate communication among people, among machines, To do this, an ontology needs to describe a formal conceptualization within a particular domain which does three things: [14].

1. Provides a semantic representation of each entity and its relationships to other entities;
2. Provides constraints and rules that permit reasoning within the ontology;
3. Describes behavior associated with stated or inferred facts.

Ontology-Based Computing. Simply introducing languages is not enough. We need an ontology-based computing infrastructure that includes: ontology development tools, content creation systems, storage and retrieval systems, ontology reasoning and mediation, and lastly, integration of reasoning with real-world applications! For preliminary work on development of ontology tools, see references [12, 18, 19, 34]. Ontologies that will enable application interoperability by resolving semantic clashes between application domains and standards/design codes are currently in development [9, 20].

For this vision to become practical, ontology-based technology must be scalable. This means that issues associated with the "expressiveness of description logic" must be balanced against "tractability of computation." While the syntax of first-order logic is designed to make it easy to say "things about objects," predicting the solution time for evaluation of statements written in standard first-order logic is often impossible. Description logics (DLs), on the other hand, emphasize "categories, their definitions, and relations," and are designed specifically for tractability of inference [7]. Description logics ensure that subsumption testing (inference) can be solved in polynomial time with respect to the size of the problem description.

DAML+OIL. DAML is an acronym for DARPA Agent Markup Language. DAML+OIL is an semantic/ontology language that ties information on a web page to machine readable semantics (ontology). An ontology consists of a set of axioms that assert resources are instances of DAML+OIL classes. which can describe the structure of a domain using the formal rigor of a very expressive description logic (DL). DAML+OIL classes can be names (URIs) or expressions (a variety of constructors are provided for building class expressions). Thus, from an implementation standpoint, a DAML+OIL ontology is a web page containing:

1. An optional daml:Ontology instance,
2. A set of classes,
3. A set of properties, and
4. A set of restrictions relating the classes and properties [10].

4.1.5 Logic (and Rules)

From this point on, we're discussing parts of the Semantic Web that are still being explored and prototyped. See Figure 11. While it's nice to have systems that understand basic semantic and ontological concepts (subclass, inverse, etc.), it would be even better if we could create logical statements (rules) that allow the computer to make inferences and deductions. Horrocks [17] points out that reasoning can be useful at many stages of the design, maintenance and deployment of ontologies:

1. Reasoning can be used to support ontology design and to improve the quality of the resulting ontology. For example, class consistency and subsumption reasoning can be used to check for logically inconsistent classes and (possibly unexpected) implicit subsumption relationships. Support of this type is particularly important for large ontologies constructed by multiple authors (just like design codes, huh?).
2. Like information integration, ontology integration can also be supported by reasoning. For example, integration can be performed using inter-ontology assertions specifying relationships between classes and properties, with reasoning being used to compute the integrated hierarchy and to highlight any problems and/or inconsistencies.

Reasoning with respect to deployed ontologies will enhance "intelligent agents" allowing them to determine, for example, if a set of facts is consistent with respect to an ontology, to identify individuals that are implicitly members of given class, and so forth. To facilitate and enable this capability, the RuleML Initiative is working towards an XML-based markup language that permits Web-based rule storage, interchange, retrieval, and firing/application. RuleML [41] encompasses a hierarchy of rules, from reaction rules (event-condition-action rules), via integrity constraint rules (consistency-maintenance rules) and derivation rules (implicational-inference rules), to facts (premiseless derivation rules).

4.1.6 Proof, Trust, and Beyond

Because the Semantic Web is an open and distributed system, in principle, anybody can say anything about anybody. To deal with the inevitable situation of unreliable and contradictory statements (data and information) on the Semantic Web, there needs to be a mechanism where we can verify that the original source does make a particular statement (proof) and that source is trustworthy (trust). At this point, notions of proof and trust have yet to be formalized, and a theory that integrates them into inference engines of the Semantic Web have yet to be developed. However, these advances in technology will occur, simply because they are a prerequisite to the building of real commercial applications.

The ability to “prove things” on the Semantic Web stems directly from its support for logical reasoning. When this system is operational, different people all around the World will write logic statements. Then, machines will follow these Semantic “links” to begin to prove facts. Swartz and Hendler [36] point out that while it is very difficult to create these proofs (it could require following thousands, or perhaps millions of the links in the Semantic Web), it’s very easy to check them. In this way, we begin to build a Web of information processors. Some of them could merely provide data for others to use. Others would be smarter, and could use this data to build rules. The smartest would be heuristic engines, powering “intelligent agents” which follow all these rules and statements to draw conclusions, and place their results back on the Web as proofs as well as data or query answers like those shown in the introduction [21].

Digital Signatures. Based on work in mathematics and cryptography, digital signatures provide proof that a certain person wrote (or agrees with) a document or statement. With a document signed (e.g., an RDF statement), a computer program agent can check if information really comes from the source it claims to be (or at least vouch for their authenticity). To ensure confidentiality of information, other security technologies like encryption and access control can also be employed.

Direct and Indirect Models of Trust. With digital signature technology in place, all that we have to do is simply tell our programs whose signatures to trust and whose not to. This simplified model works for small-scale systems that are closed. Since it is unrealistic to define the extent of trust for each source, a mechanism is necessary to derive the degree of trust for each new source. One solution is the so-called “web of trust.” When one trusts a source A, he also trusts all other sources that are trusted by source A, but to a lower extent or degree. The result is a huge and hierarchical network where agents infer information based on their trusted knowledge.

Networks of Agents. The last step in the goal of the Semantic Web is to build software agents which know about logic. These agents, with the support of the ontology, can then use RDF to navigate the sea of XML documents and perform logical reasoning tasks on behalf of a user. Each agents will probably have a very limited scope. Perhaps an agent knows how to find available times at the doctor’s office for an appointment. A second agent may know how to find available times in your personal schedule. A third agent may know how to ask the other two for available times and find a common one. A fourth agent may know how to tell agents 5 and 6 to add the appointment the doctor's schedule and your personal calendar. The key to the inference services is not in a very complex agent, but an army of simple agents who can use the Semantic Web infrastructure to communicate.

5 Requirements Engineering and the Semantic Web

The requirements engineering work breakdown structure graphically describes needs and activities. The Semantic Web Layer Cake is the graphical representation of the W3C's design solution. Establishing connections between these two models will reveal many key opportunities.

The connections from the work breakdown structure point to the minimum layer which can support that need for two reasons. First, the layer cake is a downward compatible hierarchical structure. If a particular level can address the issue, then all the layers above it can also. Therefore, it is important to identify the minimum level of semantic web technology which is needed to support the requirements engineering need. The second reason is that the upper layers are still evolving. Figure 11 illustrates the technical maturity of the semantic web layer cake. Because the map connects to the minimum layer of the semantic layer cake, it can be used to manage the development activities by focusing on the application of mature technologies and the research of evolving technologies.

5.1 Mapping the Work Breakdown Structure to the Semantic Layer Cake

Figure 16 is a map of the "Creation" branch of the work breakdown structure to the semantic web layer cake. The elicitation process will need the use of XML to support the thick descriptions. Analysis and Verification, being inference services, will need to reach the logical layer. The classification of the information will need at least the RDF layer. To become portable and reusable, it will need to be pushed up into the ontology layer. The maintenance activity, assuming the support of the classification activity, is really only concerned with integrating the various models with the design process. This can be done at the XML layer.

Figure 17 continues the mapping on the “Uses” side of the work breakdown structure.

The ability to support logical search will naturally require the logical level of the semantic web layer cake. The ability to build in the various abstraction views would benefit greatly from the logical level, however, they can be implemented by RDF models. The last element, information synthesis, will require the logical layer of the cake. This map now provides a vehicle to lay the foundation for a requirements engineering system based upon the technology of the semantic web.

6 Semantic Web Based Requirements Engineering System

A Semantic Web based requirements engineering system derives its name from the well defined, open, and universal Semantic Web standards which are its core. These standards enable creation of highly customized tools while maintaining the integrity and portability of the data. Figure 18 below depicts a graphical representation of this system.

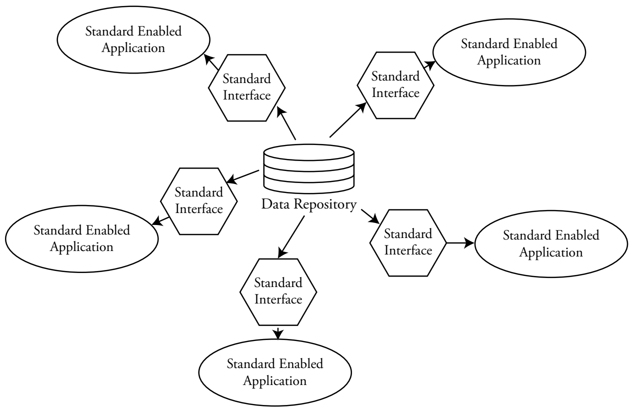

Standard Enabled Application
Standard Interface
Standard Interface
Standard Enabled Application
Standard Interface
Data Repository
Standard Enabled Application
Standard Interface
Standard Enabled Application
Standard Interface
Standard Enabled Application

Figure 18: Semantic Web Based Requirements Engineering System Deployment Vision

The data repository icon in the center is abstract. It could be a single file, a folder of files, a relational database, or a sea of information as the web exists as a sea of HTML. In the beginning there will most likely be a database manager. As the standards evolve and spread, they will replace it. Around the edges are custom applications which provide users with access points into and out of the system, via the standards.

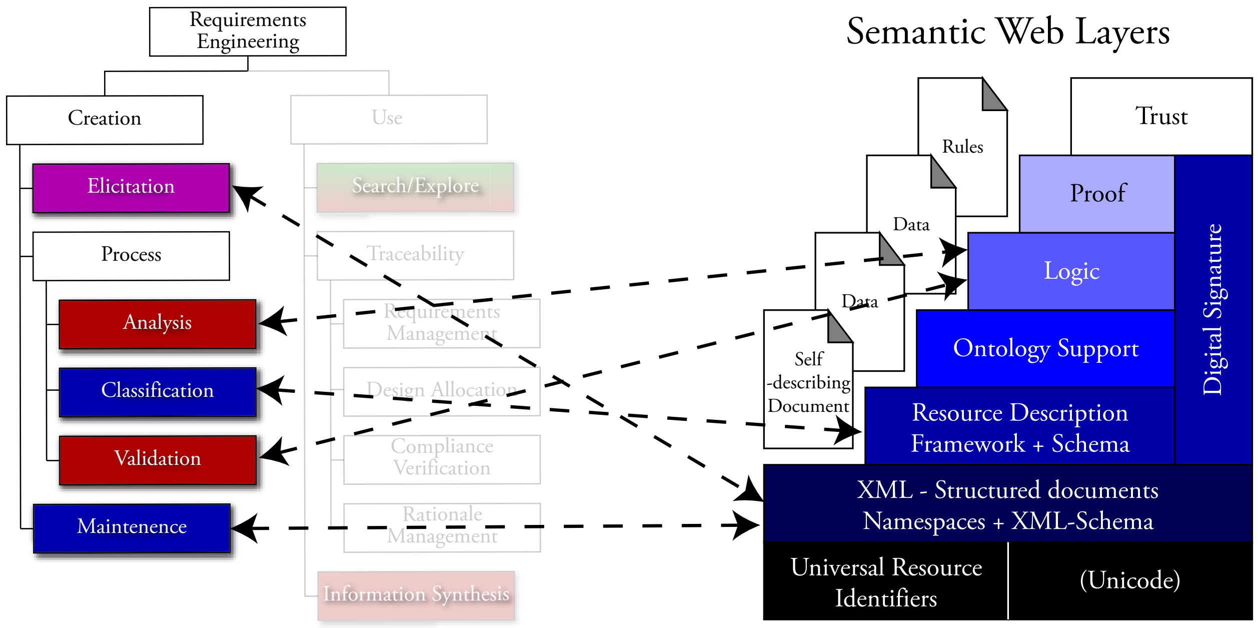


Figure 16: Mapping of Requirements Engineering Work Breakdown (Creation Branch)to the Semantic Web Layer Cake

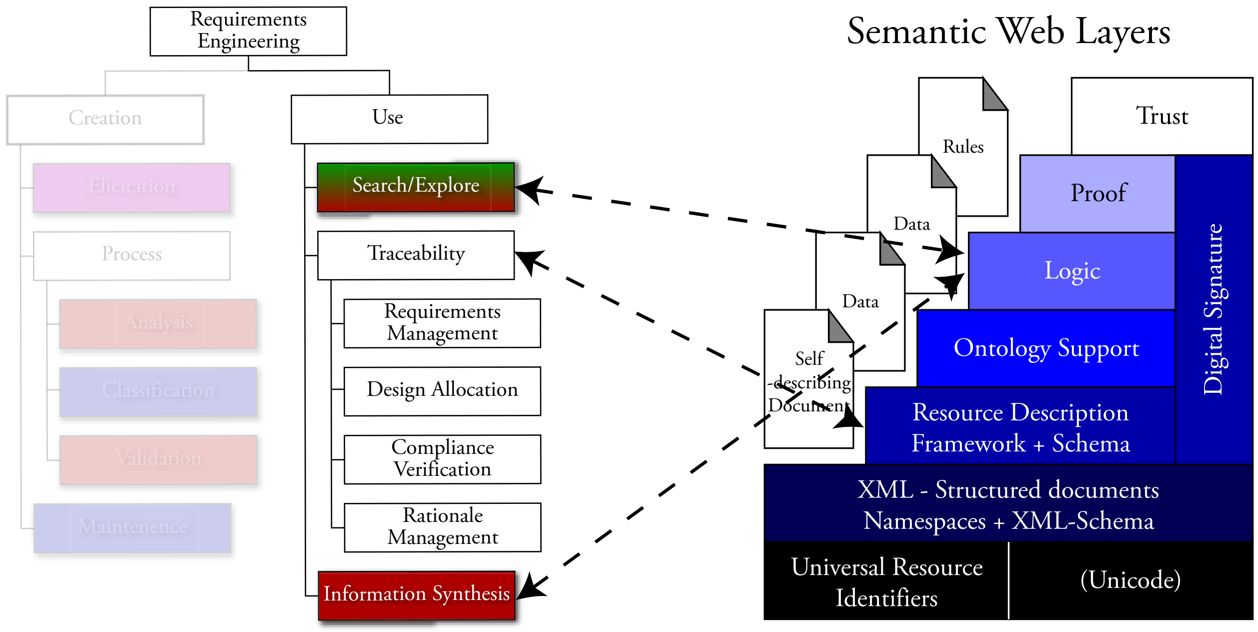


Figure 17: Mapping of Requirements Engineering Work Breakdown (Use Branch)to the Semantic Web Layer Cake

$$r = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \qquad (1)$$

```
<?xml version="1.0">
<math display ='block'
   xmlns = 'http://www.w3.org/1998/Math/MathML'>
   <mrow>
      <mi> r </mi> <mo> = </mo>
      <mfrac>
          <mrow>
               ..... etc .....
          </mrow>
          <mrow>
               ..... etc .....
          </mrow>
      </mfrac>
   </mrow>
   </math>
```

Figure 19: Quadradic Equation in traditional notation and MathML notation

6.1 Creation

The creation of the requirements document is the result of three primary activities: elicitation, processing, and maintenance. According to the work breakdown structure/Semantic Layer Cake map, XML will be able to address the needs of elicitation and maintenance. The processing activity will need the RDF layer for classification and the logical layer for analysis and verification (see Figures 16 and 17). As the web community is still searching for effective methods of realizing the ontology and logical layers, the details of analysis and verification will be deferred.

A critical need of the elicitation process is to capture the requirements with all the rich and thick descriptions. XML is well suited to this task. Not only can XML support textual description, but it can also support more complex information formats such as equations as shown with MathML[25] in Figure 19 and graphics (Scalable Vector Graphics language [35]). Being portable, XML will support the desire to reuse information.

Keeping the requirements engineering process in step with the design process is the primary goal of maintenance. There are two keys to making this step a reality. The first is to define models identifying the necessary information and structure. A useful analogy for models of this type is to compare them with forms. Forms specify required fields, optional fields, and determine valid parameters within each field. The second key is to link these models into the design environment. This is mostly a useability concern. The requirements engine interface could be developed as a separate and isolated application; however, in practice the task of maintaining the data will be dropped to keep to the project schedule. The data will then become incomplete and untrustworthy, and therefore useless.

6.1.1 XML as a Bridging Technology

The task of creating the models will be discussed later under the activity of classification. The present discussion will focus directly on the integration of the models into the design process. The first hurdle is to find a way to bridge information between applications. It cannot be expected that any two applications will have knowledge of the other or their personal data models. XML is an excellent answer to this problem. It is an open standard with public domain parsers readily available. This has opened the door for software developers to build XML ready applications. Several major software vendors, such as Adobe Software and Microsoft, have already incorporated XML into their flagship applications. In Figure 20 below we extend the model of Figure 18 by adding XML as the standard between the applications.

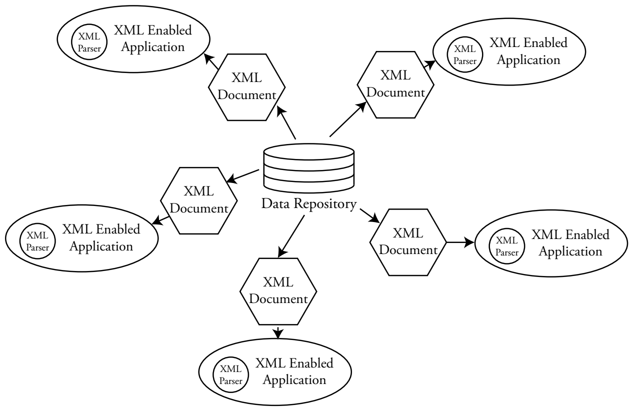


Figure 20: XML Portability Model

A simple example, a requirement on the battery life of a cell phone, will demonstrate the feasibility of this model. The requirement reads as follows: an active (in-use) cell phone shall last at least 2 hours on a full battery charge. This information will be needed by the internal lab documentation, the manufacturing quality assurance test, and an advertising brochure. The model for the information pathway is pictured below in Figure 21.

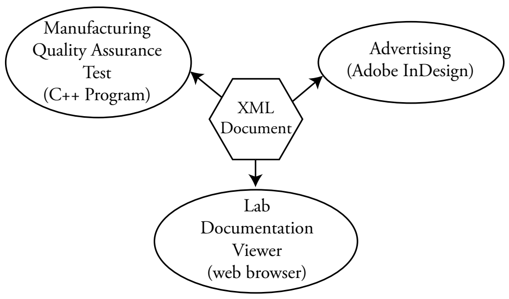


Figure 21: Requirement Information Pathway

Here's the base xml file which describes this requirement:

```
<?xml version="1.0"?>
<requirement>
   <description>
      An active (in-use) cell phone shall last at least <value unit="hour"> 2 </value>
      hours on a full battery charge
   </description>
</requirement>
```

The simplest way to display the information as part of the internal lab documentation is with an XML enabled web browser and a cascading style sheet. In the following example, note the inclusion of the stylesheet reference in the XML document.

cellphone.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="cellphone.css"?>
<requirement>
   <description>
      An active (in-use) cell phone
      shall last at least
      <value unit="hour"> 2 </value>
      hours on a full battery charge
   </description>
</requirement>
```

cellphone.css

```
value {font-size:18pt; color:red;}
```

The result of viewing this XML document in a XML ready browser are shown in Figure 22.

The manufacturing test code will be simulated by a piece of Java code which reads the file, and compares a uniformly distributed random variable from one to three, to the desired performance metric. The

Java code includes the Xerces [43] open source parser for reading the XML. This program is shown in Figure 23. Finally, the XML document is imported into Adobe InDesign 2.0, an industry standard application for professional publications. See Figure 24.

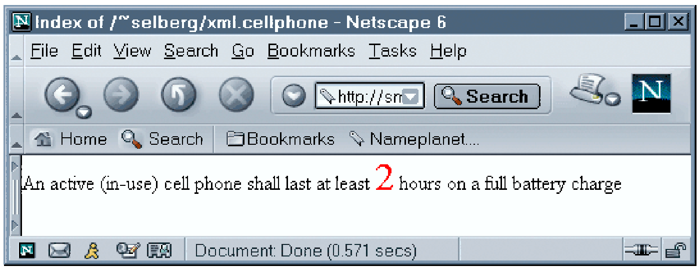
If we change the XML file, then reload it into the various applications we get the results illustrated in Figures 25, 26 and 27.

6.1.2 Critical Examination of XML – Need for XSLT

In this simple case, we can use XML to pass information between applications; the bridge is built. But let us look a bit more closely and see how tenuous the thread is that ties these applications together. The XML standard allows the application to read the data file, but it does not provide any guarantee that the application can interpret it. An XML document created by MS Word, while containing all the data, will most likely be completely incompatible with Adobe InDesign [2].

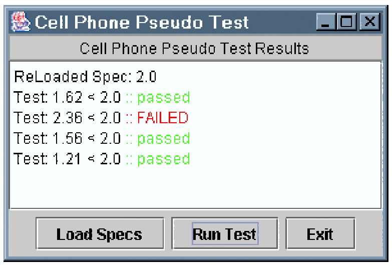
There is, however, a mechanism around this problem: transformations. XSLT [37, 45] is the eXtensible Stylesheet Language for Transformations. It is a standard by which a style sheet can be created and applied to an XML document which has the power to restructure or relabel the data. Figure 28 illustrates this concept. An initial XML file conforms to a certain document model. This is illustrated by the hexagonal shape of the XML document and the negative hexagonal shape of the DTD/Schema template. This document is then passed into a transformation engine, for example James Clark's XT [46], with a complementary style sheet. The transformation engine uses the style sheet to map the initial document into a new model. This is illustrated by the triangle shaped XML document.

To highlight the power of this concept, two examples will be introduced. The first transforms the earlier cell phone example into a more stylized and portable HTML. The second style sheet will perform a common engineering task by changing the units of a requirement from hours to minutes. Internet Explorer 5 and Netscape 6/7 both support XML with XSLT transformations internally. The demonstration model is shown in Figure 29. Note that the XML transformation engine has been specified as XSLT and the final application is now a web browser.


Index of /~selberg/xml.cellphone - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
http://sm
Search
Home
Search
Bookmarks
Nameplanet...
Document: Done (0.571 secs)

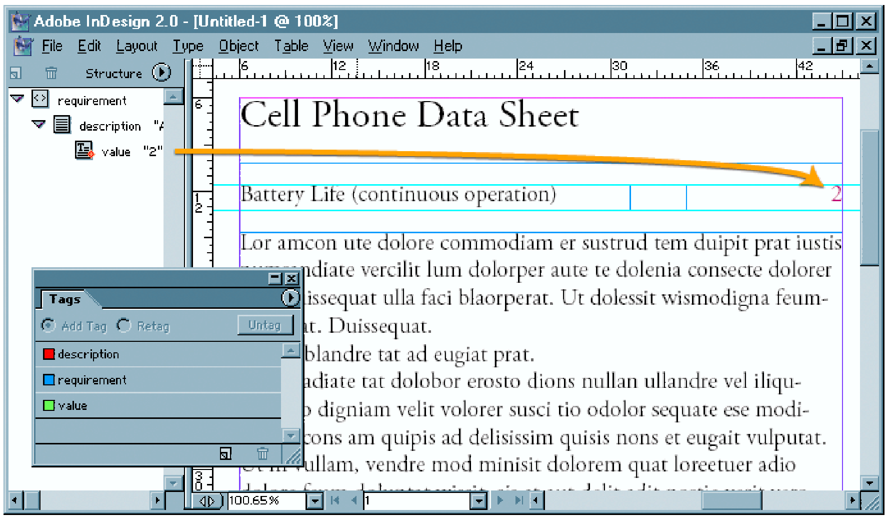
An active (in-use) cell phone shall last at least 2 hours on a full battery charge

Figure 22: Viewing an XML File in Netscape with a Cascading Style Sheet



```
ReLoaded Spec: 2.0
Test: 1.62 < 2.0 :: passed
Test: 2.36 < 2.0 :: FAILED
Test: 1.56 < 2.0 :: passed
Test: 1.21 < 2.0 :: passed
```

Figure 23: Test Program Written in Java, which uses the XML Data Source

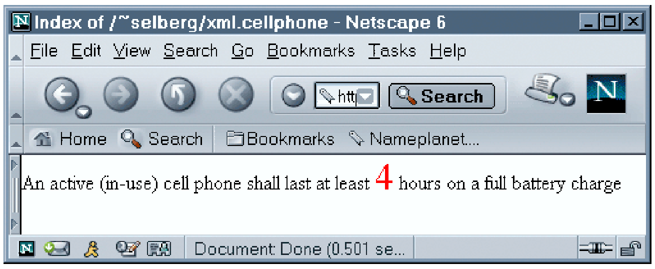


Cell Phone Data Sheet

Battery Life (continuous operation) 2

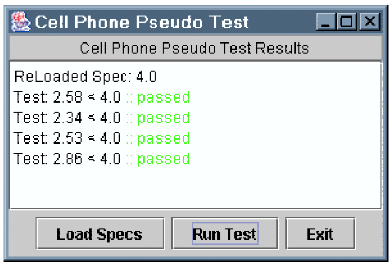
Lor amcon ute dolore commodiam er sustrud tem duipit prat iustis
diate vercilit lum dolorper aute te dolenia consecte dolorer
issequat ulla faci blaorperat. Ut dolessit wismodigna feum-
t. Duissequat.
blandre tat ad eugiat prat.
adiate tat dolobor erosto dions nullan ullandre vel iliqu-
digniam velit volorer susci tio odolor sequate ese modi-
cons am quipis ad delisissim quisis nons et eugait vulputat.
ullam, vendre mod minisit dolorem quat loreetuer adio

Figure 24: Importing the XML Data File into Adobe InDesign 2.0.


Index of /~selberg/xml.cellphone - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
Search
Home Search Bookmarks Nameplanet...
Document: Done (0.501 se...

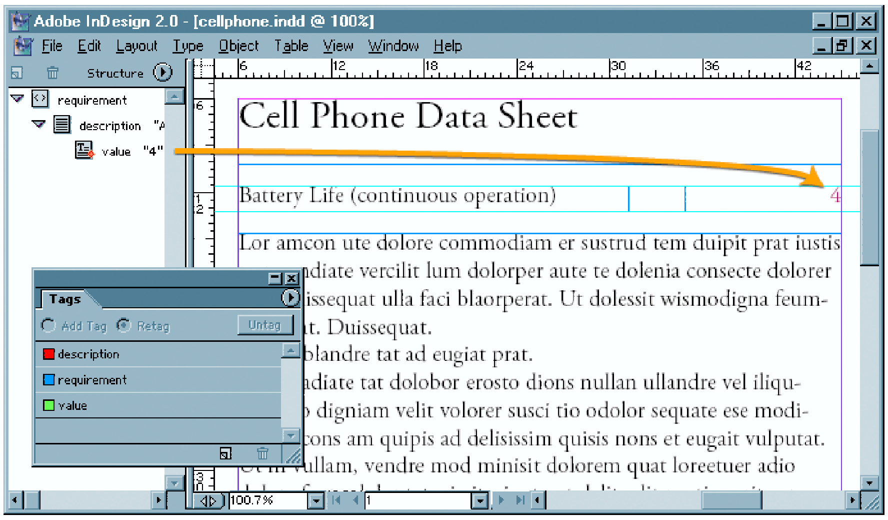
An active (in-use) cell phone shall last at least 4 hours on a full battery charge

Figure 25: Netscape View of the Modified XML file



```
ReLoaded Spec: 4.0
Test: 2.58 < 4.0 : passed
Test: 2.34 < 4.0 : passed
Test: 2.53 < 4.0 : passed
Test: 2.86 < 4.0 : passed
```

Figure 26: Java Test View of the Modified XML file



Cell Phone Data Sheet

Battery Life (continuous operation) 4

Lor amcon ute dolore commodiam er sustrud tem duipit prat iustis
ndiate vercilit lum dolorper aute te dolenia consecte dolorer
issequat ulla faci blaorperat. Ut dolessit wismodigna feum-
t. Duissequat.
blandre tat ad eugiat prat.
ndiate tat dolobor erosto dions nullan ullandre vel iliqu-
o digniam velit volorer susci tio odolor sequate ese modi-
cons am quipis ad delisissim quisis nons et eugait vulputat.
ullam, vendre mod minisit dolorem quat loreetuer adio

Figure 27: Indesign View of the Modified XML file

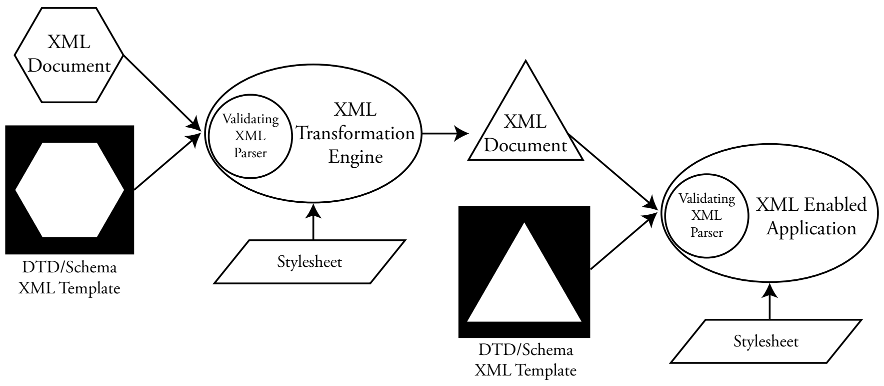


Figure 28: XML Transformation Process

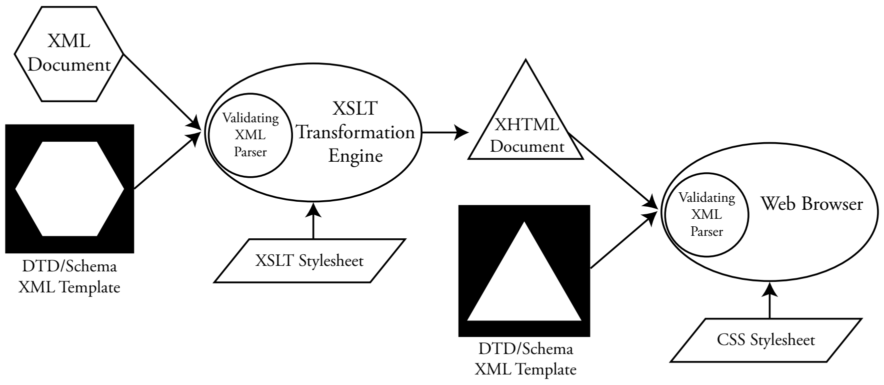


Figure 29: XML to XHTML Transformation Process

Since the XSLT engine is already built into the browser, the process will appear to be a single step. However, Figure 30 shows that the model is the same with the exception that the inner step has been automated.

The initial XML file, which is passed to the browser, has a reference to the corresponding XSLT style sheet. The style sheet has been customized to the tags in the XML document and provides the templates into which the tags will be transformed. The most important number in this example is the value. The style sheet lets the engine know that when it sees a <description> tag (see cellphone.xsl, line 18), it should first call the ShowValue template (see cellphone.xsl, line 19 and 26),which extracts and displays the value. The transformation engine then returns back to where it was (see cellphone.xsl, line 20), and continues to process the information. This provides a automatic way of extracting the key information.

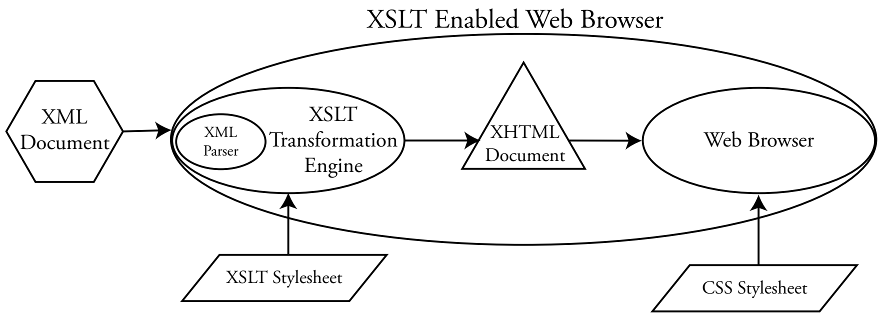


Figure 30: XML/XSLT Enabled Web Browser

cellphone.xml

```
<?xml version="1.0">;
<?xml-stylesheet type="text/xsl" href="cellphone.xsl"?>
<requirement>
   <description>
      An active (in-use) cell phone shall last at
      least <value unit="hour"> 2 </value> hours
      on a full battery charge.
   </description>
</requirement>
```

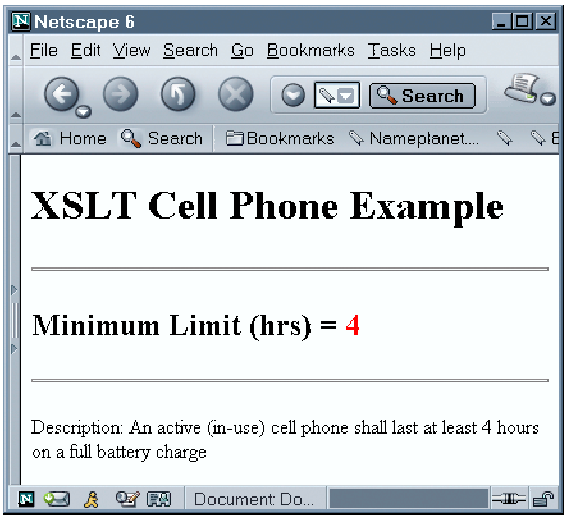
cellphone.css

```
span.value { font-size:18pt; color:red; }
```

The results are in cellphone.xsl

```
<?xml version="1.0">;
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="html"/>
<xsl:template match="requirement">
   <html>
      <head>
         <title> XSL Cellphone Stylesheet Example </title>
         <link rel="stylesheet" type="ext/css" href="cellphone.t.css"/>
      </head>
      <body>
         <h1> XSLT Cell Phone Example&lt; </h1> </hr
         <xsl:apply-templates/>
      </body>
   </html>
</xsl:template>

<xsl:template match="description">
   <xsl:call-template name="ShowValue">
   <p>
      <xsl:text> Description: </xsl:text>
      <xsl:apply-templates>
   </p>
<xsl:template>

<xsl:template match="ShowValue">
   <h2>
      <xsl:text> Minimum Limit (hours) = </xsl:text>
      <span class="value">
         <xsl:value-of select="value/text()"/>
      </span>
   </h2>
   <hr/>
</xsl:template>

</xsl:stylesheet>
```

The graphical rendering of cellphone.xsl is shown in Figure 31.


Netscape 6
File Edit View Search Go Bookmarks Tasks Help
Search
Home Search Bookmarks Nameplanet...
Document: Do...

XSLT Cell Phone Example

Minimum Limit (hrs) = 4

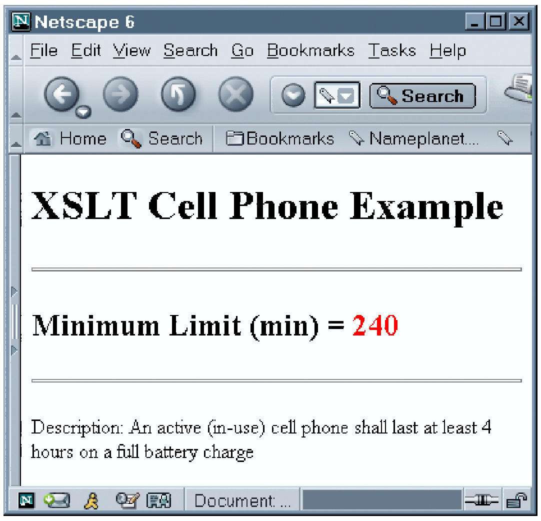
Description: An active (in-use) cell phone shall last at least 4 hours on a full battery charge

Figure 31: XSLT Transformation from XML to XHTML

With a slight modification, the ShowValue template can transform not just the display formatting, but also the value. In the original XML file, the units attribute within the value tag was set to hours (see, cellphone.xml, line 6). The stylesheet can use this attribute to automatically transform the value to minutes. This ability is important because it provides a mechanism of allowing both the writer and the user to use personally convenient units without making every application aware of every unit and conversion. The abbreviated stylesheet and results of this transformation are summarized below in the scripts of XML code.

```
<xsl:template name="ShowValue">
<h2> <xsl:text> Minimum Limit </xsl:text>
<xsl:choose>
   <xsl:when test="contains( value/@unit, 'hour' ) ">
   <xsl:text> (min) = </xsl:text>
      <span class="value">
         <xsl:value-of select="value/text() * 60.0"?>
      </span>
   </xsl:when>
   <xsl:otherwise>  <xsl:text> ( </xsl:text>
                                 <xsl:value-of select="value/@unit/text()">
                                 <xsl:text>) = </xsl:text>
                    <span class="value">
                          <xsl:value-of select="value/text()"/>
                    </span>
   </xsl:otherwise>
</xsl:choose>
</h2>
<hr/>
</xsl:template>

</xsl:stylesheet>
```

The modified value (240 seconds) is shown in Figure 32.


Netscape 6
File Edit View Search Go Bookmarks Tasks Help
Search
Home Search Bookmarks Nameplanet...
Document: ...

XSLT Cell Phone Example

Minimum Limit (min) = 240

Description: An active (in-use) cell phone shall last at least 4 hours on a full battery charge

Figure 32: Unit Transformation from Hours to Minutes through XML and XSLT

Transformations provide the ability to re-purpose information in an automated way. This is of critical value to the industry because it provides protection against obsolescence. As the ontologies and agents evolve, transformations provide a method for continually realigning the information. XML provides the ability to capture thick data and transport is across applications. XML also supports the creation of custom data models through the use of DTDs and Schema; however, there is not yet support for validating custom document templates within standard industry tools. Therefore, XML can bridge the data across applications, but it can not yet bridge the data models.

6.1.3 Using RDF to Relate Information

Classifying the data will take more than XML. The first required extensions are the various data models which define the classification. For example, the traceability reference models introduced earlier. The reference models, being graphs rather than trees, will need to be implemented in RDF. In addition to needing to model graphs, there is a need to merge domain specific classification models. For example, a cell phone base station design will need to integrate geographical information, population density information, and broadcast power information while developing area coverage requirements and specifications. While it is conceivable to build an industry standard super-model encompassing all the domains, it is more practically realistic to expect each domain to develop and build a distinct and unique model, which will later be merged with others. Since graphs can be merged into a combined graph, it is an appropriate structure for building the data models. This again points to the need for RDF. Given a classification model which fits the graph model, it is rather trivial to serialize the data into RDF/XML.

To see how these issues might work in practice, consider the need to store a decision. The rationale requirements traceability sub-model of Ramesh and Jarke [31], shown in Figure 6 provides a classification model. Now consider the following hypothetical scenario:

> It is unclear which data format to use for building a requirements engineering system. The options include XML and Microsoft Word [42]. Among the criteria for the decision was a need for cross application portability, thus XML was selected.

The graph of this decision object is shown below in Figure 33. The object is labeled "XMLDecision" is the target of the first rdf:Description element identifying it as the subject. Beneath that element are the various property arcs, such as "based_on" and "resolve". Within each arc element is another rdf:Description element to mark the object node. The third arc, "select," points to a node, which in turn points to two other nodes. This demonstrates the flexibility of RDF. While the translation of the RDF into serialized XML is straightforward, it is rather tedious as shown in below. Tools are needed to encode the data models and provide user-friendly interconnections between the graphs and the RDF/XML representations.

```
<?xml version="1.0">
```

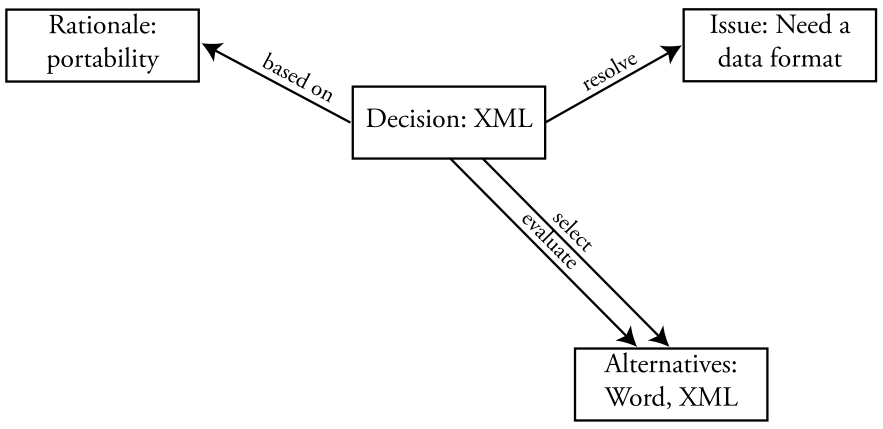


Figure 33: Graphical Representation of a Decision

```
<object type=decision id=XMLDecision>
   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax/ns#">
      <rdf:Description about="XMLDecision">
         <based_on>
            <rdf:Description about="Rationale"/>
                                 <member>
                                             <rdf:Description about="portability"/>
                                      </member>
                              </rdf:Description>
         </based_on>
         <resolve>
            <rdf:Description about="Issue"/>
                                 <member>
                                             <rdf:Description about="NeedADataFormat"/>
                                      </member>
                              </rdf:Description>
         </resolve>
         <select>
            <rdf:Description about="alternatives">
               <member>
                  <rdf:Description about="Word"/>
                  <rdf:Description about="XML"/>
               </member>
            </rdf:Description>
         </select>
      </rdf:Description>
   </rdf:RDF>
</object>
```

6.2 Use

Usage of the requirements engineering system can be broken out into three activities. First is a need to abstract the information (i.e. see the forest as opposed to the trees). Implementation will require the development of an abstraction model. The model can be instantiated through RDF and XML. The second application is to explore the traceability threads. The number of ways the system can use the traceability information is infinite. Much of the power of the system is the ability to build customized tools to work with the data in personalized ways. An example of such a system is provided in the next section. The last activity concerns inference services. Inference services include verification, validation, logical search, inference, and consequence exploration. At this point in time, the Semantic Web community is still actively researching an infrastructure which would support these activities. They will be critically tied to ontologies, which is another active area of research. Therefore, the discussion of how the requirements system will adopt them will need to be deferred to a later date.

7 An XML/RDF Traceability Viewer

XML and RDF introduce a new and profound ability to interleave and link information; thus they are an excellent foundation for a traceability viewer. In this example, requirements, design descriptions, and tests, have been developed within an XML/RDF framework. The end viewer is Internet Explorer 6. Internet Explorer was chosen because the Adobe scalable vector graphics (SVG) plugin, which is used to view the xml-based vector graphics, is not yet fully compatible with Netscape 6.0/7.0 or Mozilla 1.0 [28, 29, 35]. The XML translation language, XSLT, provides the interface between the XML, RDF and the browser.

7.1 The Example

To demonstrate the viewer, an example case is needed. To that end a basic electronic device, the metronome, will be modeled. A metronome produces an audible click at a frequency specified by the user. It is used by musicians as a training tool. A photo of such a device is shown below in Figure 34.

A simplified requirement tree for this device is described in Figure 35. The top level requirement is for a time reference. Refining this requirement are sub requirements for an adjustable frequency and an audible tone (the user input and output). The user needs control of the volume, so a volume control is

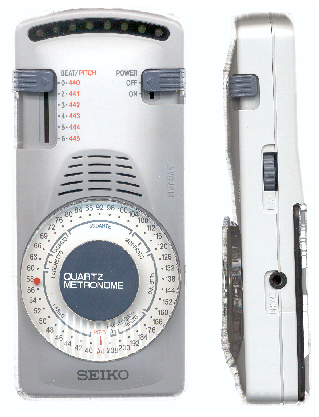


Figure 34: Front and Side Elevation Views of Metronome

required. The frequency requirement is broken down into a user interface and a range requirement.

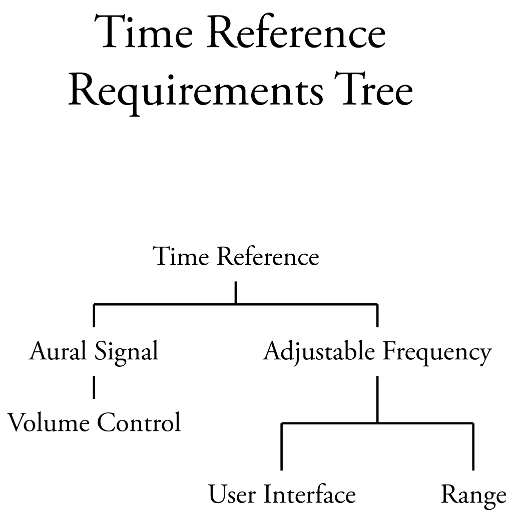


Figure 35: Tree of Time Reference Requirements

A simplified block diagram of the device is given below in Figure 36. The metronome is broken down into three groups: the amplitude group, the frequency group, and the power group. The amplitude group consists of an variable amplifier and a control knob. The frequency group is composed of a variable oscillator and a control knob. Both the amplitude and frequency groups have direct relationships to the requirements. The power group is a derived group from the choice of electronics. It is not directly determined by the user requirements, but rather by the design requirements. The power supply has an on/off switch. The power then fans out to the various active system components (ie the oscillator and the amplifier) through a distribution amplifier.

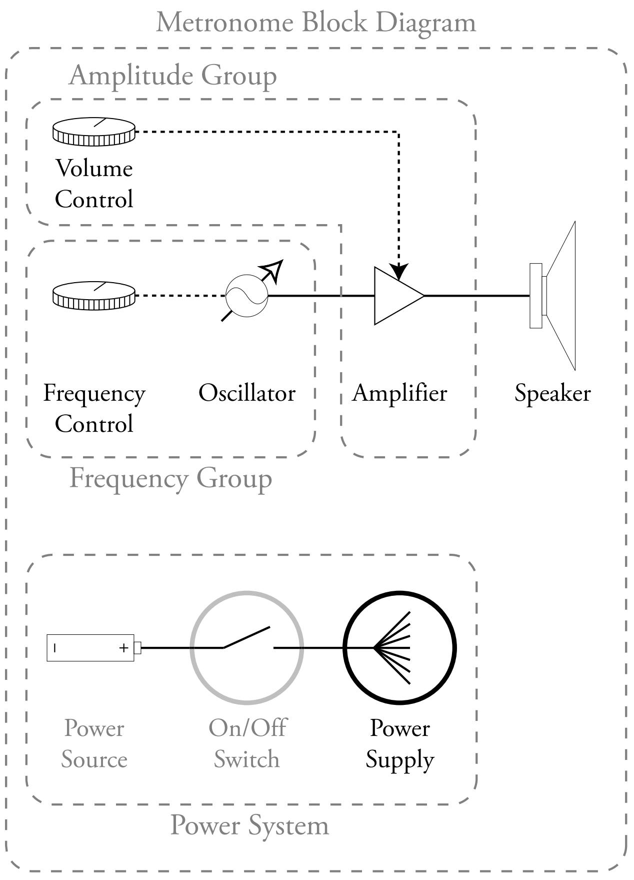


Figure 36: Metronome Block Diagram

7.2 The Framework

To build a traceability browser for this example, a traceability model is needed. The model which will be used is described in Figure 37. It is a variation on the low-end traceability reference model from Ramesh and Jarke [31].

Each of the core blocks listed will have several additional data fields. These fields will not be explicitly called out for brevity's sake. The major distinction between the original model and this model is the explicit inclusion of bidirectional references.

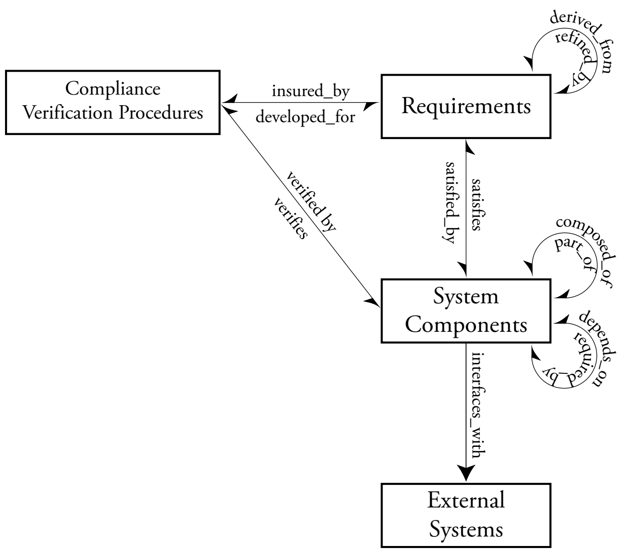


Figure 37: Metronome Example Traceability Model

7.3 Building the XML Requirement Files

The basic format of the requirement, component, and test files are identical. The variance derives from a few element names as defined by Figure 37. As an example, the top level requirement, Time Reference, is shown below.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="requirement.xsl"?>

<requirement id="TimeReference.req.xml"
             xmlns:="http://www.isr.umd.edu/~selberg"
             xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

   <description>
      A physical device which displays a signal at regular frequency.
   </description>

   <need>
       For a performing musician it can often be difficult to sustain a constant
       beat.  This is especially true when the music is complex, or very slow
       and sustained.  A physical time reference allows the musician to train
       his or her internal clock to a higher degree of precision.
   </need>

   <rdf:RDF>
   <rdf:Description about="TimeReference.req.xml">
      <refined_by>
         <rdf:Description about="AuditorySignal.req.xml"/>
         <rdf:Description about="AdjustableFrequency.req.xml"/>
      </refined_by>
      <satisfied_by>
```

```
            <rdf:Description about="Metronome.com.xml"/>
        </satisfied_by>
    </rdf:Description>
    </rdf:RDF>

    <history>
        <entry timestamp="12:38:03 PM 3/2/02"> Created Requirement </entry>
    </history>
</requirement>
```

The <rdf:RDF> block is the core of the linking. The first rdf:Description block identifies the subject of the RDF statement. Beneath it are two property arcs: refined_by and satisfied_by. These arcs then point to the subject objects.

7.4 Requirements Viewer

The requirements viewer takes advantage of a number of Internet technologies, organized as shown in Figure 38. The implementation being described has three key object types: Requirements, Tests, and Components. Each type has a specific accompanying XML structure. Within each definition is a reference to the desired stylesheet which allows the internet browser to find and apply the stylesheet. The XSLT stylesheet contains the information necessary to transform the XML and RDF into linked HTML. During the transformation two scalable vector graphics (SVG) files, some javascript, and a cascading style sheet reference become embedded or linked into the HTML page. The two SVG files provide an abstraction mechanism for navigating the requirements and components, respectively. The javascript code provides interactivity within the browser to facilitate the navigation and traceability interrelationships. Lastly, the cascading style sheets provide some simple formatting.

The complete stylesheet, which transforms the XML requirement document into HTML is rather lengthly; however, it is essentially many variations on a single theme. The style sheet sets up the HTML template with links to the SVG files, javascript code, and the cascading style sheet. For most of the fields in the XML file, the stylesheet places a heading and then lists out the text. The following script shows a typical XSLT template for viewing an ordinary element, such as the description.

```
<?xml version="1.0">
```

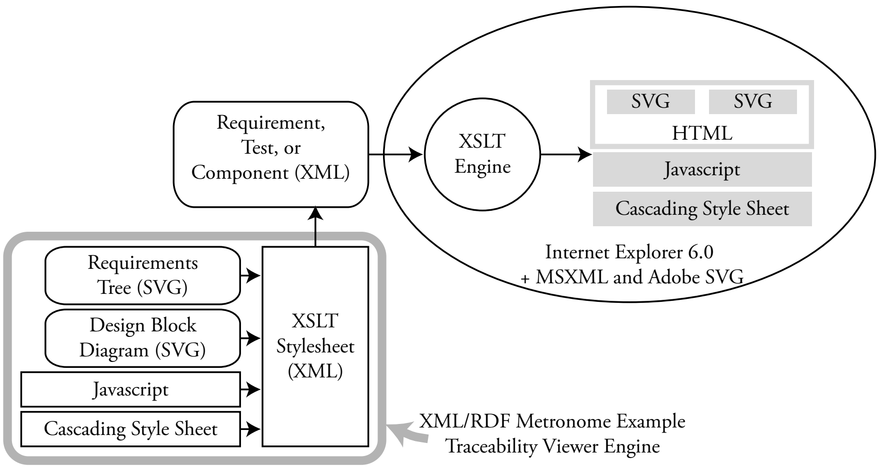
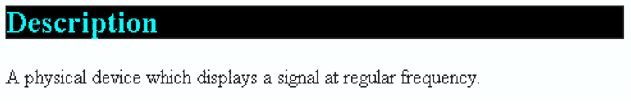

Requirement, Test, or Component (XML)
XSLT Engine
SVG
SVG
HTML
Javascript
Cascading Style Sheet
Internet Explorer 6.0 + MSXML and Adobe SVG
Requirements Tree (SVG)
Design Block Diagram (SVG)
Javascript
Cascading Style Sheet
XSLT Stylesheet (XML)
XML/RDF Metronome Example Traceability Viewer Engine

Figure 38: Model of the Example XML/RDF Traceability Viewer

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<xsl:template match= "description">
   <h2> Description </h2>
   <p class = "description" >
      <xsl:value-of select="child::text()"/>
   </p>
<xsl:template>
</xsl:stylesheet>
```

The fragment of XML code below describes the metronome and is rendered in an Internet browser, as shown in Figure 39. (The color formatting was done through an applied cascading style sheet)

```
<description>
   A physical device which displays a signal at regular frequency.
</description>
```



Description

A physical device which displays a signal at regular frequency.

Figure 39: Description Viewed in an Internet Browser

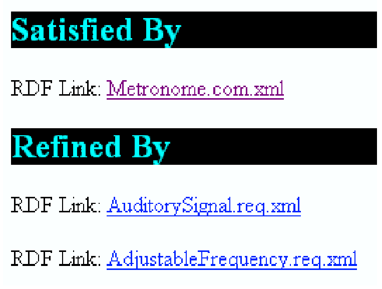
The processing of RDF links requires a more elaborate transformation. First, we observe that the XSLT style sheet knows each of the various RDF arc types. When the style sheet discovers an **`rdf:Description`** element, it recursively calls the rdfDescription template. This template compares the element name against all of the known rdf arc types. When it finds a match it scans for the object node.

The template prints out a the arc name as a title in the h2 format. Underneath it prints out the object nodes as HTML links. The abbreviated stylesheet is as follows:

```
<?xml version="1.0"?>
<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

   <xsl:template name="rdfDescription" >
   <xsl:param name="rdfPlace"/>
   <xsl:param name="rdfFirst"/>
   <xsl:param name="rdfSecond"/>

       <xsl:if test="verified_by">
           <h2>Verified By</h2>

           <xsl:for-each select="verified_by">
           <xsl:if test="text()">
               <p class="verified_by">
                  <xsl:value-of select="text()"/>
               </p>
           </xsl:if>

           <xsl:for-each select="rdf:Description">
               <xsl:call-template name="rdfDescription">
               <xsl:with-param name="rdfPlace"
                               select="concat($rdfPlace,':=:verified_by:=:',@about)"/>
               <xsl:with-param name="rdfFirst"
                               select="$rdfPlace"/>
               <xsl:with-param name="rdfSecond"
                               select="@about"/>
               </xsl:call-template>
           </xsl:for-each>
           </xsl:for-each>
       </xsl:if>

       <xsl:if test="verifies">
                 <h2>Verifies</h2>

                 <xsl:for-each select="verifies">
                 <xsl:if test="text()">
                     <p class="verifies">
                        <xsl:value-of select="text()"/>
                     </p>
                 </xsl:if>

                 <xsl:for-each select="rdf:Description">
                     <xsl:call-template name="rdfDescription">
                     <xsl:with-param name="rdfPlace"
                               select="concat($rdfPlace,':=:verifies:=:',@about)"/>
                     <xsl:with-param name="rdfFirst"
                               select="$rdfPlace"/>
                     <xsl:with-param name="rdfSecond"
                               select="@about"/>
                     </xsl:call-template>
                 </xsl:for-each>
                 </xsl:for-each>
       </xsl:if>

      ..... stylesheet cut ......

       <xsl:if test="@about">
```

```
            <xsl:if test="../../part_of |
                          ../../composed_of |
                          ../../depends_on |
                          ../../required_by |
                          ../../refined_by |
                          ../../derived_from |
                          ../../satisfies |
                          ../../satisfied_by |
                          ../../insured_by |
                          ../../verifies |
                          ../../verified_by">
                  <p> RDF Link: <a href="{@about}"
                          id="{$rdfPlace}"
                          onmouseover="on_mouse_over('{$rdfSecond}')"
                          onmouseout="on_mouse_out('{$rdfSecond}')">
                          <xsl:value-of select="@about"/></a></p>
            </xsl:if>
        </xsl:if>

        <xsl:if test="text()">
            <p> RDF Attribute: <xsl:value-of select="text()"/> </p>
        </xsl:if>
    </xsl:template>
</xsl:stylesheet>
```

Figure 40 shows the stylesheet viewed in an Internet browser. The use of RDF provides a clear mechanism for setting up the recursion within the style sheet, thus it is easy to write parsing routines which can be generalized and reused for other RDF applications. The complete top level requirement is shown in Figure 41.


Satisfied By

RDF Link: Metronome.com.xml

Refined By

RDF Link: AuditorySignal.req.xml

RDF Link: AdjustableFrequency.req.xml

Figure 40: RDF Link Results Viewed in an Internet Browser

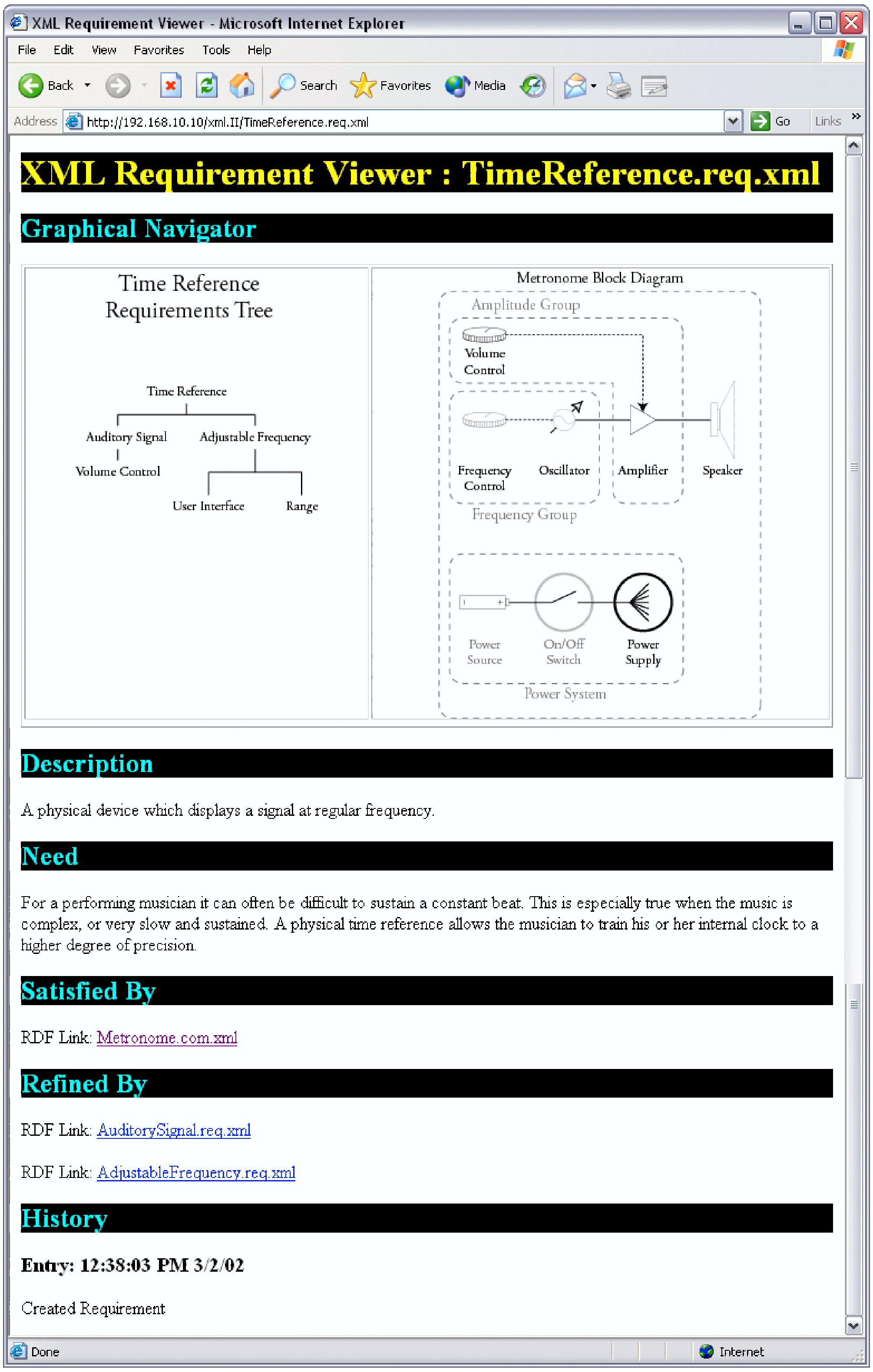
7.5 Viewer User Interactivity

The real power of having XML behind the scenes is about to reveal itself. The javascript code is able to read the XML/RDF block and build interactivity between the displayed RDF links and the embedded scalable vector graphic (SVG) documents. By "mousingover" the word "TimeReference" in the requirements tree graphic, all of the references which directly link to the requirement become highlighted with color coding. This is shown below in Figure 42.

The red coloring on word "TimeReference" indicates that the current view is linked to that item. Green items indicate requirements which are refined_by or derived_from. A complementary pattern is green coloring for components which are part_of or composed_of. The magenta coloring indicates components which satisfy requirements, or requirements which are satisfied_by components. By "mousingover" the word "Auditory Signal" in the requirements tree, or the link to AuditorySignal.req.xml, only the signal requirement and the "TimeReference" remain highlighted. This is because "mousingover" causes only directly related items, as defined by the XML document being viewed, to be highlighted. This situation is shown in Figure 43.

Clicking on either reference to the Auditory Signal will allow the browser to move to that requirement, as shown in Figure 44. In Figure 45, the final part of our demonstration, we "mouseover" the Auditory Signal word in the requirement tree. This has the effect of turning on all on all highlighting. The SVG graphics have not changed; the highlighting is a function of the mouse state and the XML file being viewed. Thus we can see that the Amplitude Group is the design aspect which satisfies the auditory signal requirement. We have just taken a step down a traceability path.

In this example we have demonstrated many of the key features for a requirements traceability system. XML has been applied as the technology for capturing requirements. While the information captured in this example was extremely simple, it has already been shown that XML can encapsulate rather complex information. The RDF, SVG, and Javascript extensions have allowed the instantiation of an interactive abstraction model to assist the user in navigating and understand the traceability relationships. This example focused on a viewer, thus it did not show any ability to apply models to the capture and maintenance of requirements; however, it should be noted that XML supports DOM and Schema for applying a model to the information. Thus this example demonstrate how the semantic web technologies is providing immediate answers to some of the basic needs of requirements engineering, while promising future solutions for others.



XML Requirement Viewer - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://192.168.10.10/xml.II/TimeReference.req.xml

XML Requirement Viewer : TimeReference.req.xml

Graphical Navigator

Description

A physical device which displays a signal at regular frequency.

Need

For a performing musician it can often be difficult to sustain a constant beat. This is especially true when the music is complex, or very slow and sustained. A physical time reference allows the musician to train his or her internal clock to a higher degree of precision.

Satisfied By

RDF Link: Metronome.com.xml

Refined By

RDF Link: AuditorySignal.req.xml

RDF Link: AdjustableFrequency.req.xml

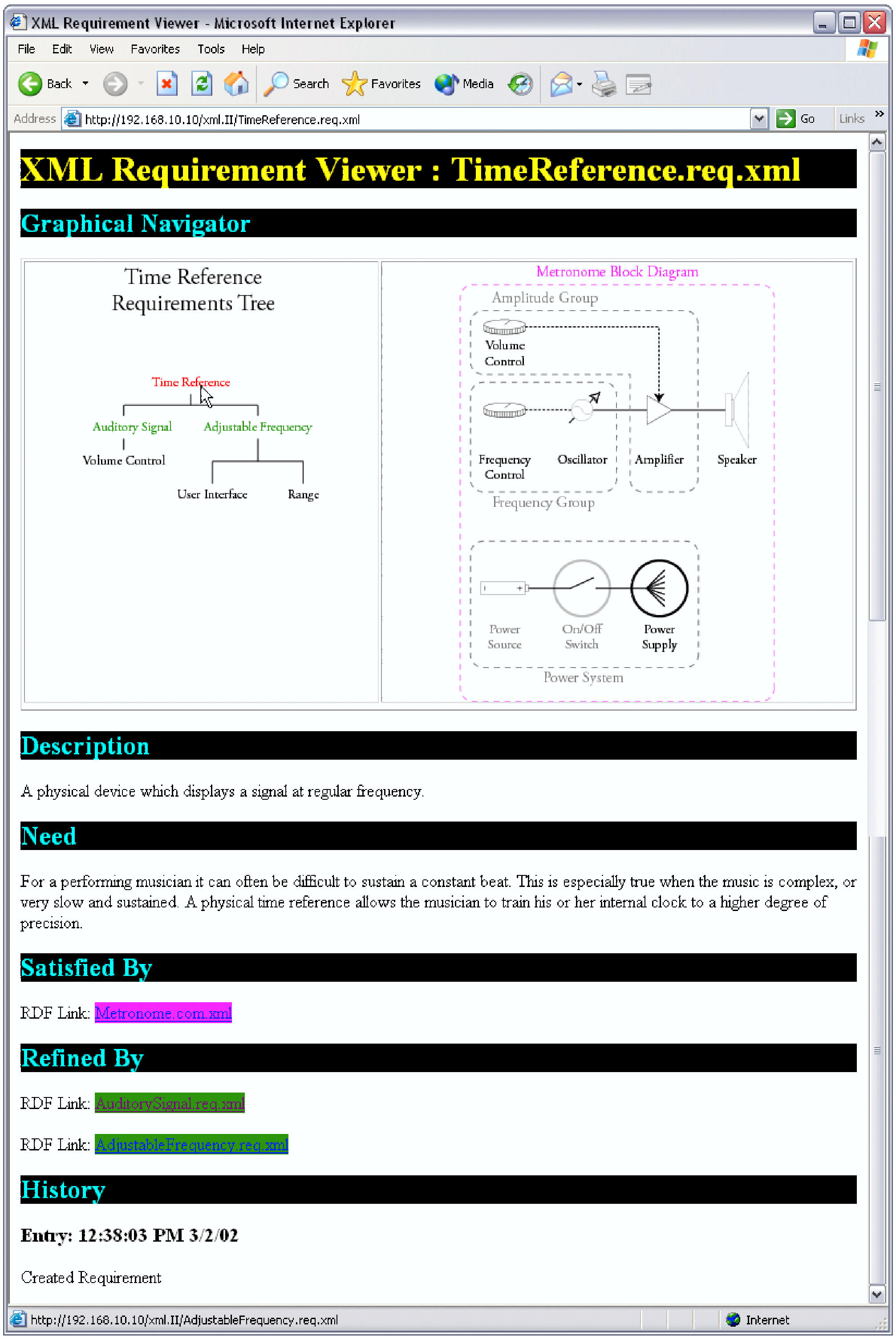
History

Entry: 12:38:03 PM 3/2/02

Created Requirement

Done Internet

Figure 41: Interactive Traceability Browser. Rendering of Time Reference Requirement


XML Requirement Viewer - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://192.168.10.10/xml.II/TimeReference.req.xml Go Links
Time Reference Requirements Tree
Time Reference
Auditory Signal
Adjustable Frequency
Volume Control
User Interface
Range
Metronome Block Diagram
Amplitude Group
Volume Control
Frequency Control
Oscillator
Amplifier
Speaker
Frequency Group
Power Source
On/Off Switch
Power Supply
Power System
http://192.168.10.10/xml.II/AdjustableFrequency.req.xml
Internet

XML Requirement Viewer : TimeReference.req.xml

Graphical Navigator

Description

A physical device which displays a signal at regular frequency.

Need

For a performing musician it can often be difficult to sustain a constant beat. This is especially true when the music is complex, or very slow and sustained. A physical time reference allows the musician to train his or her internal clock to a higher degree of precision.

Satisfied By

RDF Link: Metronome.com.xml

Refined By

RDF Link: AuditorySignal.req.xml

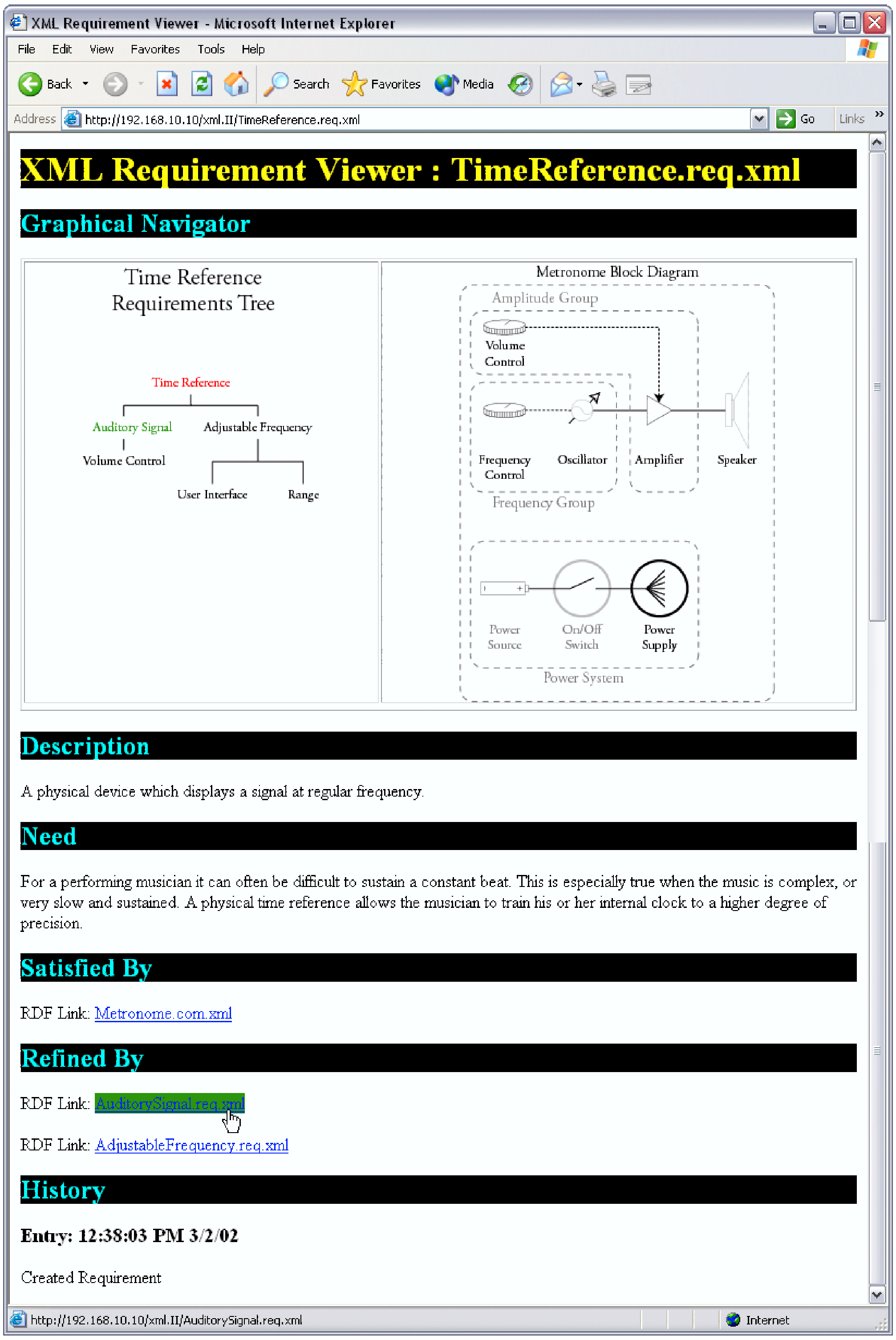
RDF Link: AdjustableFrequency.req.xml

History

Entry: 12:38:03 PM 3/2/02

Created Requirement

Figure 42: Interactive Traceability Browser. When we "mouseover" the Time Reference Requirement, all of the connecting requirements are highlighted



XML Requirement Viewer : TimeReference.req.xml

Graphical Navigator

Description

A physical device which displays a signal at regular frequency.

Need

For a performing musician it can often be difficult to sustain a constant beat. This is especially true when the music is complex, or very slow and sustained. A physical time reference allows the musician to train his or her internal clock to a higher degree of precision.

Satisfied By

RDF Link: Metronome.com.xml

Refined By

RDF Link: AuditorySignal.req.xml

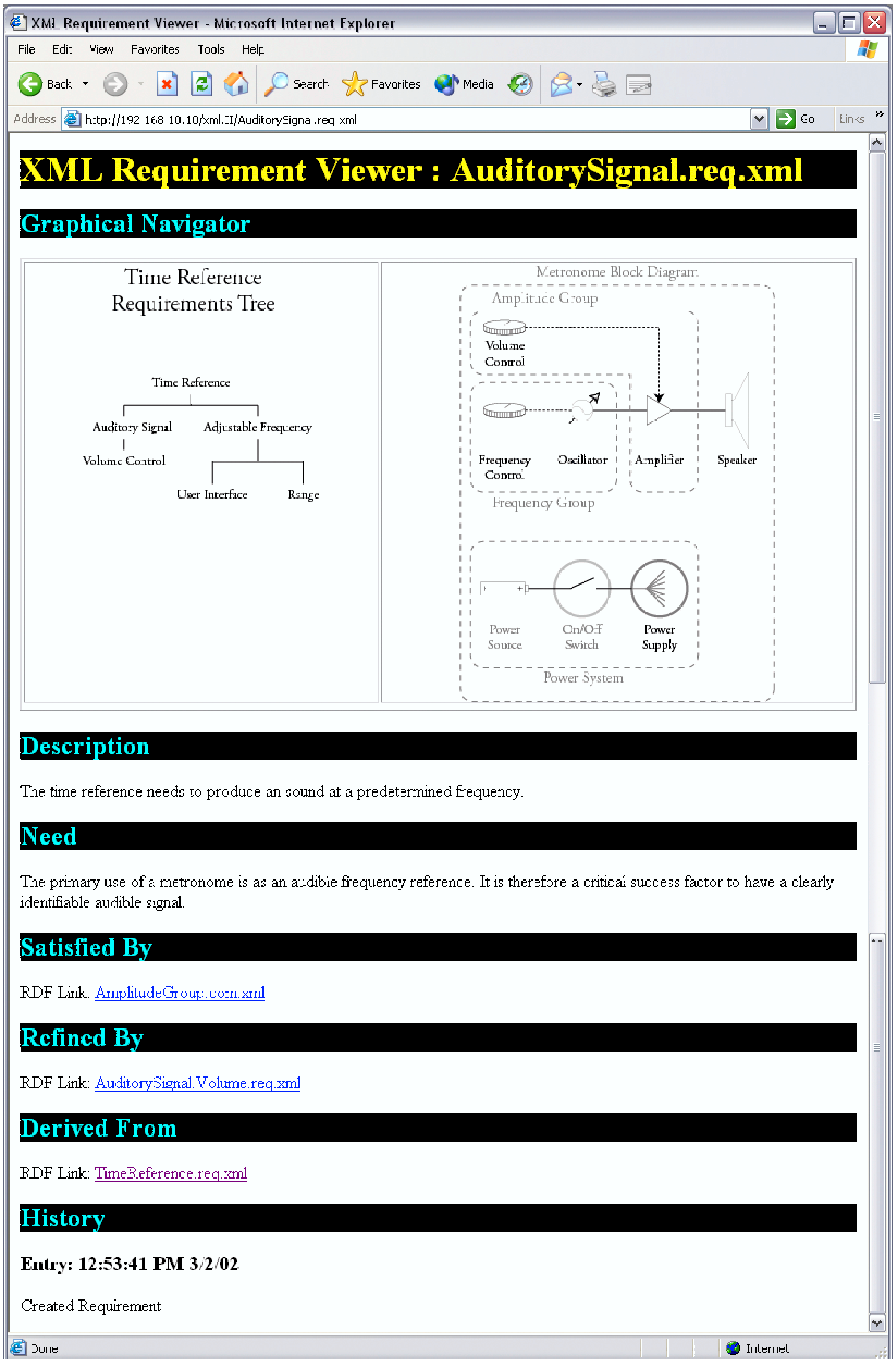
RDF Link: AdjustableFrequency.req.xml

History

Entry: 12:38:03 PM 3/2/02

Created Requirement

Figure 43: Interactive Traceability Browser. When we "mouseover" the Auditory Signal reference, only the Auditory Signal and Time Reference requirements remain highlighted (in top left-hand corner box)



XML Requirement Viewer : AuditorySignal.req.xml

Graphical Navigator

Description

The time reference needs to produce an sound at a predetermined frequency.

Need

The primary use of a metronome is as an audible frequency reference. It is therefore a critical success factor to have a clearly identifiable audible signal.

Satisfied By

RDF Link: AmplitudeGroup.com.xml

Refined By

RDF Link: AuditorySignal.Volume.req.xml

Derived From

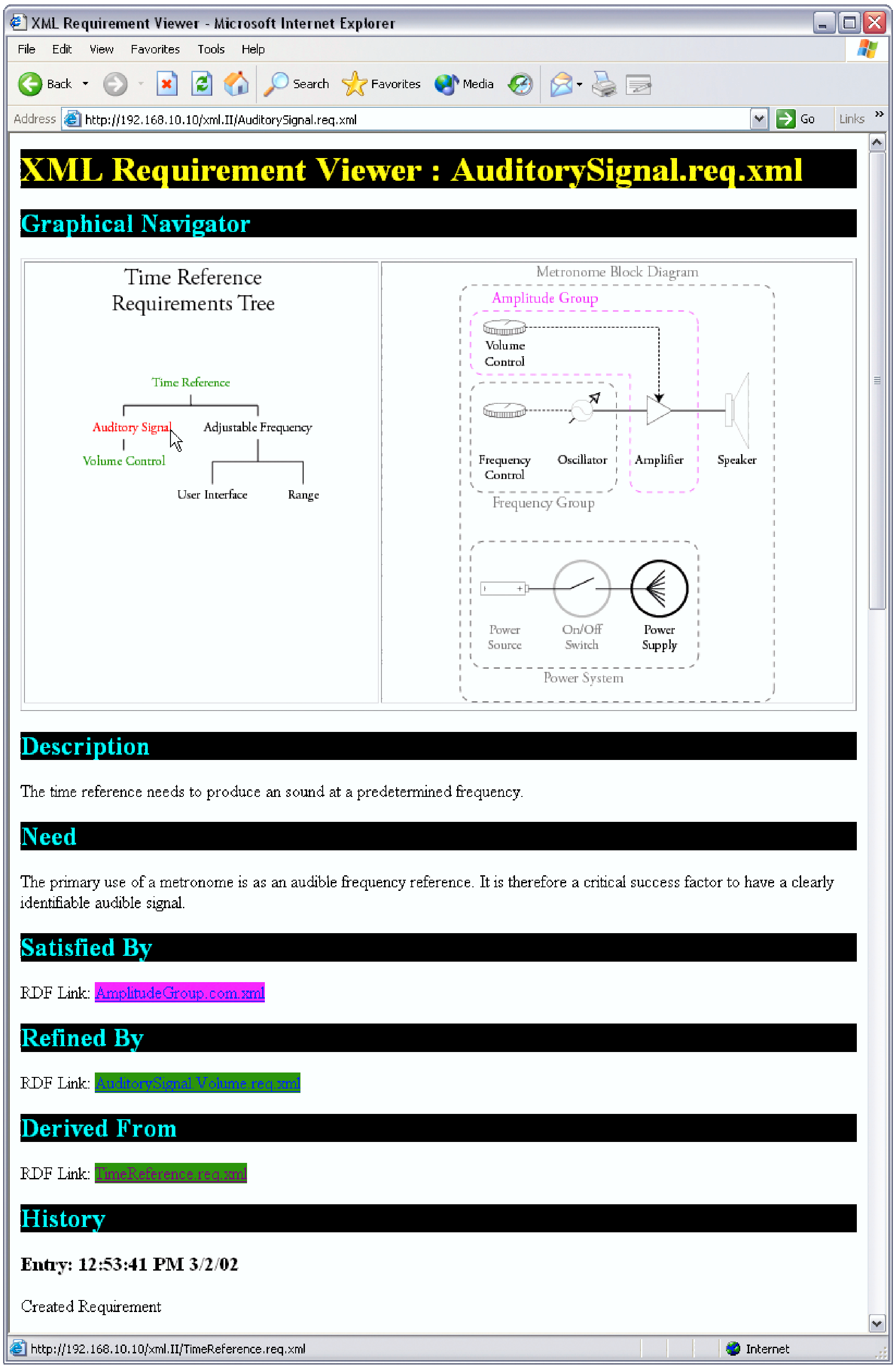
RDF Link: TimeReference.req.xml

History

Entry: 12:53:41 PM 3/2/02

Created Requirement

Figure 44: Interactive Traceability Browser. Rendering of Auditory Signal Requirement



XML Requirement Viewer : AuditorySignal.req.xml

Graphical Navigator

Description

The time reference needs to produce an sound at a predetermined frequency.

Need

The primary use of a metronome is as an audible frequency reference. It is therefore a critical success factor to have a clearly identifiable audible signal.

Satisfied By

RDF Link: AmplitudeGroup.com.xml

Refined By

RDF Link: AuditorySignal.Volume.req.xml

Derived From

RDF Link: TimeReference.req.xml

History

Entry: 12:53:41 PM 3/2/02

Created Requirement

Figure 45: Interactive Traceability Browser. When we "mouseover" the Auditory Signal Requirement, traceability links to the "satisfied by," "refined by" and "derived from" requirements are highlighted.

8 Conclusions and Future Work

8.1 Conclusions

A key observation of this work is the identification of a requirements engineering system as a system of systems. From this observation comes the conclusion that the appropriate design strategy is to focus around standards and interfaces rather than a suite of complex applications. The contribution of this work is to synthesize a connection between requirements engineering and the Semantic Web standards. This approach has immediate advantages because the Semantic Web either has a technology, or is actively researching a technology, which will address each of the major present day deficiencies identified by Ramesh and Jarke [31]. In the long run, the decision to base the system on industry accepted standards opens up massive possibilities for improved economics of development, through leverage and reuse. The four deficiencies and the complementary web technology are listed below.

1. **Need for Thick Descriptions.**The current industry tools do not easily support the inclusion of implicit rich, thick descriptions of requirements, but prefer the more computer friendly "thin" descriptions. XML can be used to simultaneously represent thick and thin descriptions.
2. **Need for Model-Driven Trace and Capture.** Two aspects of this issue need to be addressed. The first is an ability to define custom models. With these models in place, the second task is to integrate them into the design applications. RDF, XML, and the corresponding schema allow for the implementation of models. Industry acceptance of XML is allowing it to become a bridging technology, thus permitting the requirements engineering system to integrate itself into the mainstream applications. There are not yet general support mechanism for integrating the document models, or schema.
3. **Need for Abstraction Mechanisms.** There is a need to build requirements engineering tools that can manipulate and view requirements at multiple levels of abstraction. In the closing example (See Section 7), RDF and XML were used to implement an abstraction model. The success of this implementation suggests that XML and RDF are capable of instantiating much more complex models. However, because a general reference model has not yet been researched and developed for requirements,

a definitive conclusion cannot be made.

4. **Need for Inference Services.** Inference services are automated routines to perform or assist in the activities of verification, validation, logical search, inference, and consequence exploration. As engineering projects grow in scope, scale, and duration, the need for these services becomes critical. Within the Semantic Web community the development of inference services is an active area of research. It is therefore not possible to make positive conclusions about them at this time. However, tying the requirements system to the Semantic Web opens up the door to a large research community. Thus the answers may be years closer.

The opportunities for improved methods of leverage and reuse are a direct consequence of building the system around open and well defined standards. The appropriateness of the design approach is a direct consequence of identifying both the Internet and the requirements engineering system as chaotic systems of systems. They are systems of systems because each can be decomposed into independent and intelligent systems. The chaotic label refers to the lack of a central management structure. In these design domains, the interfaces, and thus the standards, are everything. The standards provide the pathway for a company to package it's specifications and requirements, and pass them along to other companies to be reused.

In figuring out how "requirements engineering" methodologies and tools can benefit from Semantic Web technologies, it is likely that we have to deal to with many trade-offs. First, present-day search engines are capable of finding answers to searches that cover a huge part of the Internet; however, there is no notion of correctness in the answers. At the same time, present-day logic engines have the capability of restricting their output to information that is known to be correct, but are currently incapable of wandering through masses of intertwined data to construct answers. The process of dealing with the combinatorial explosion of possibilities that need to be traced is quite intractable. A key question is: can we get the best of both worlds and apply these techniques to a new generation of "requirements engineering" tools?

8.2 Future Work

The field of possibilities for future work is enormous. To give some structure to the opportunities, it is useful to construct a requirements engineering layer cake which describes the infrastructure necessary for a requirements engineering system. The layer cake is diagrammed in Figure 46.

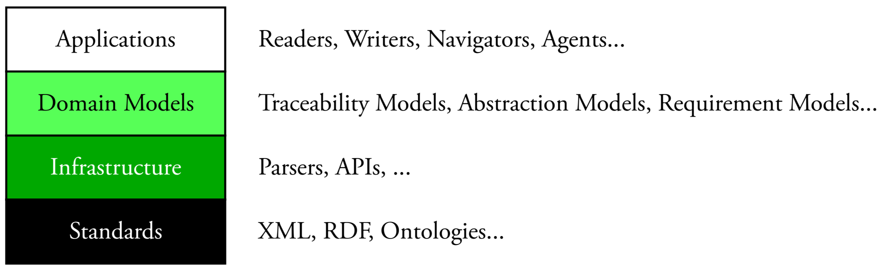

Applications
Readers, Writers, Navigators, Agents...
Domain Models
Traceability Models, Abstraction Models, Requirement Models...
Infrastructure
Parsers, APIs, ...
Standards
XML, RDF, Ontologies...

Figure 46: Requirements Engineering Layer Cake

The bottom two layers of this cake are borrowed from the Semantic Web. The standards layer contains the open and widely accepted standards such as XML. The infrastructure layer refers to the tools needed to work with the standards. For example, in building a house, if nails are the connection standard, then a hammer is needed. If XML is the data format, then code modules to read and write XML are needed.

The "domain models" layer deals with the specific models necessary to support requirements engineering. Ramesh and Jarke [31] have provided an excellent traceability reference model and a process for instantiating it. Similar work should be conducted to create reference abstraction models, requirements models, decision models, and so forth. These models can then be implemented using the Semantic Web standards. The application layer is last. Presently, it is empty except for a few simple examples. Thus, there is a vast field of opportunity to develop tools which can use the system architecture to achieve productivity results beyond anything possible today. For example, corporations today need to monitor 200,000+ environmental regulations and requirements on a monthly basis [33]. This task is too large to be cost effectively managed by people alone. The new generation of tools will be able to monitor the requirements for changes. When a change occurs, an inference service will be able to make a decision about whether the change is relevant or not. If it is relevant, it will ask if the change is significant. If the change is relevant and significant, then inform the support team. This will greatly improve the efficiency of engineering.

While some of the borrowed elements in the standards layer are well developed and proven, the semantic web community is still actively researching aspects of it, ontologies for example. Incorporating this research will be extremely important for furthering the generality and flexibility of the requirements engineering system. One of the immediate needs of the requirements engineering system is an ability to apply document models or schema for XML and RDF within industry standard applications. At the infrastructure layer, there is a strong need for RDF tools such as readers, writers, and navigators. To support reusability of information, there is also need to merge document models. RDF, as a graph description, can support this transaction in theory; however, the ability has not yet been demonstrated. Such a system would take several RDF graphs together, with mappings among them, and integrate them into a new combined graph as shown in 47.

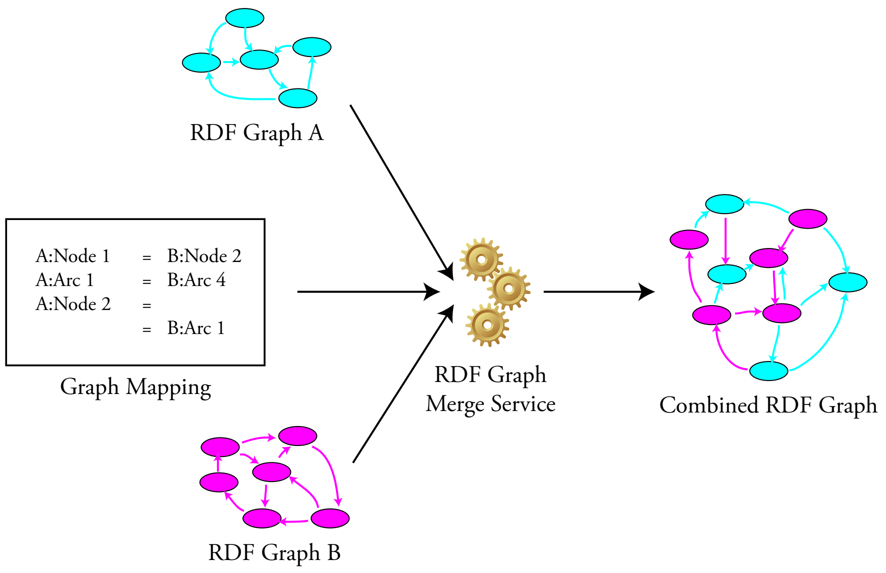


Figure 47: Combining RDF Graphs

References

[1] 2002. Adobe Illustrator. Referenced on April 18, 2002. See http://www.adobe.com/products/illustrator/main.html.

[2] 2002. Adobe InDesign. Referenced on April 18, 2002. http://www.adobe.com/products/indesign/main.html.

[3] AP233 Property-Based Requirements ARM Models, Version Alpha-1, July 2002. Project Leader: Jim U'Ren (juren@jpl.nasa.gov).

[4] AP233 Text-Based Requirements ARM Models, Version Beta-1, June 2002. Project Leader: Jim U'Ren (juren@jpl.nasa.gov).

[5] Austin, M.A. Lecture Notes for ENSE 622/ENPM 642: Systems Engineering Requirements, Design and Trade-Off Analysis, 2002.

[6] Austin, M.A., and Frankpitt B.A. Lecture Notes for ENSE 621 and ENPM 641 Systems Engineering Principles, 2000.

[7] Baader F., Calvanese D., McGuinness D., Nardi D., Patel-Schneider P. *The Description Logic Handbook.* Cambridge University Press, February 2003.

[8] Berners-Lee, T., 2000. XML and the Web. Keynote address at the XML World 2000 Conference.

[9] Ciocoiu M., Gruninger M., Nau D.S. Ontologies for Integrating Engineering Applications. *Journal of Computing and Information Science in Engineering*, 1(1):12–22, 2001.

[10] Fensel D., van Harmelen F., Horrocks I., McGuinness D., Patel-Schneider P. OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems*, pages 38–45, March/April 2001.

[11] Fowler M., and Scott K. *UML Distilled Second Edition.* Addison-Wesley, Reading, Massachusetts, 2000.

[12] Golbeck J., Grove M., Parsia B., Kalyanpur A., and Hendler J. New Tools for the Semantic Web. In *Proceedings of 13th International Conference on Knowledge Engineering and Knowledge Management EKAW02*, Siguenza, Spain, October 2002.

[13] Google Search Engine. Referenced on April 7, 2002., 2002. See http://www.google.com.

[14] Gruniger M., and Lee J. Ontology Applications and Design. *Communications of the ACM*, 45(2):39–41, February 2002. Referenced on April 7, 2002. http://www.google.com.

[15] Hendler J. Agents and the Semantic Web. *IEEE Intelligent Systems*, pages 30–37, March/April 2001. Available on April 4, 2002 from http://www.computer.org/intelligent.

[16] Herzog E., Torne A. Support for Representation of Functional Behavior Specifications in AP233. 2002.

[17] Horrocks I. DAML+OIL: A Description Logic for the Semantic Web. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 2001.

[18] Kalyanpur A., Golbeck J., Grove M., and Hendler J. An RDF Editor and Portal for the Semantic Web. July 2002.

[19] Kalyanpur A., Parsia B., Hendler J., Golbeck J. SMORE - Semantic Markup, Ontology, and RDF Editor, 2003. Maryland Information and Network Dynamics (MIND) Lab, University of Maryland, College Park. For details, see http://www.mindswap.org.

[20] Kiliccote H., Garrett J.H. Standards Usage Language (SUL). *Journal of Computing in Civil Engineering*, 15(2):118–128, 2001.

[21] Lu S., Dong M., and Fotouhi, F. The Semantic Web: Opportunities and Challenges for Next-Generation Web Applications. *Information Research*, 7(4), 2002. Available at: http://InformationR.net/ir/7-4/paper134.html.

[22] Maier, M.W. Architecting Principles for Systems of Systems. *Systems Engineering*, 1999.

[23] Maier, M.W., and Rechtin E. *The Art of Systems Architecting, 2nd Edition.* CRC Press, London, 2000.

[24] Manola, Frank, and Miller, Eric., 2002. RDF Primer, Available on April 4, 2002 from http://www.w3.org/TR/2002/WD-rdf-primber-20020319. Work in Progress.

[25] 2002. MathML. Referenced on April 6, 2002. See hhttp://www.w3.org/Math.

[26] Metacrawler. Referenced on April 7, 2002., 2002. See http://www.metacrawler.com.

[27] Meyer, Erik A. *Cascading Style Sheets: The Definitive Guide.* O'Reilly and Associates, Sebastopol, California, 2000.

[28] 2002. Mozilla. Referenced on April 5, 2002. See http://www.mozilla.org/projects/mathml.

[29] 2002. Netscape. Referenced on April 7, 2002. See http://www.netscape.com.

[30] Pandikow A., Torne A. Support for Object-Orientation in AP-233. 2002.

[31] Ramesh, B., and Jarke, M. Toward Reference Models for Requirements Traceability. *IEEE Transactions on Software Engineering*, 27(2):58–93, 2001. Available on April 4, 2001 from http://citeseer.nj.nec.com/ramesh99towards.html.

[32] 2002. RDF. Referenced on April 5, 2002. http://www.w3.org/RDF.

[33] Sampson M.E. Web-based Requirement Management and Regulatory Tracking. People, Teams, and Systems. 1998.

[34] Sirin E., Hendler J., Parsia B. Semi-automatic Composition of Web Services using Semantic Descriptions, 2002. Accepted to "Web Services: Modeling, Architecture and Infrastructure" Workshop in conjunction with ICEIS2003.

[35] 2002. Scalar Vector Graphics (SVG). Referenced on April 5, 2002. See http://www.w3.org/Graphics/SVG/Overview.html.

[36] Swartz A., and Hendler J. The Semantic Web: A Network of Content for the Digital City, 2002. Available at http://blogspace.com/rdf/SwartzHendler.

[37] Tidwell D. *XSLT*. O'Reilly and Associates, Sebastopol, California, 2001.

[38] 2002. Unicode. Referenced on April 4, 2002. See http://www.unicode.org.

[39] 2002. URI. Referenced on April 4, 2002. See http://www.isi.edu/in-notes/rfc239c.txt.

[40] World Wide Web Consortium (W3C). Referenced on April 4, 2002., 2002. See http://www.w3.org.

[41] Wagner G. How to Design a General Rule Markup Language. *Eindhoven University of Technology*, 2002. See http://tmitwww.tm.tue.nl/staff/gwagner.

[42] 2002. Microsoft Word, Referenced on April 5, 2002. See http://www.microsoft.com/office/word.

[43] 2002. Xerces. Referenced on April 4, 2002. See http://xml.apache.org.

[44] 2002. XHTML. Referenced on April 5, 2002. See http://www.w3.org/MarkUp.

[45] 2002. XSLT. Referenced on April 5, 2002. See http://www.w3.org/Style/XSL.

[46] 2002. XT. Referenced on April 6, 2002. See http://www.jclark.com/xml/xt.html.